Student Declaration of Authorship

HERIOT WATT UNIVERSITY
UK | DUBAI | MALAYSIA

Course code and name:	F20PA Research Methods and Requirements Engineering - 2023-2024
Type of assessment:	**Individual**
Coursework Title:	Final Year - Dissertation Project
Student Name:	Siteng Wan
Student ID Number:	H00364927

Declaration of authorship. By signing this form:

- **I declare** that the work I have submitted for individual assessment OR the work I have contributed to a group assessment, is entirely my own. I have NOT taken the ideas, writings or inventions of another person and used these as if they were my own. My submission or my contribution to a group submission is expressed in my own words. Any uses made within this work of the ideas, writings or inventions of others, or of any existing sources of information (books, journals, websites, etc.) are properly acknowledged and listed in the references and/or acknowledgements section.

- I confirm that I have read, understood and followed the University's Regulations on plagiarism as published on the University's website, and that I am aware of the penalties that I will face should I not adhere to the University Regulations.

- I confirm that I have read, understood and avoided the different types of plagiarism explained in the University guidance on Academic Integrity and Plagiarism

Student Signature *(type your name):* *Siteng Wan*

Date: *14/04/2024*

Copy this page and insert it into your coursework file in front of your title page.
For group assessment each group member must sign a separate form and all forms must be included with the group submission.

Student Declaration of Authorship

HERIOT WATT UNIVERSITY
UK | DUBAI | MALAYSIA

Your work will not be marked if a signed copy of this form is not included with your submission.

Performance of single-board computers on image processing tasks

Author
Siteng Wan

School of Mathematics and Computer Science

A thesis presented for the degree of
BEng Robotics

Supervisor
Hans Wolfgang Loidl

Heriot-Watt University
April 15, 2024

Number Pages: 66

Declaration of Authorship

I, Siteng Wan, declare that this thesis titled, 'Performance of single-board computers on image processing tasks' and thework presented in it is my own. I confirm that this work submitted for assessment is myown and is expressed in my own words. Any uses made within it of the works of otherauthors in any form (e.g., ideas, equations, figures, text, tables, programs) are properly acknowledged at any point of their use. A list of the references employed is included.

Signed: Siteng Wan

Dated: 14/4/2024

Performance of single-board computers on image processing tasks

Siteng Wan

Abstract

The motivation for this study is driven by the need to guide the selection of suitable hardware for executing vision recognition tasks on single-board computers (SBCs), particularly in resource-constrained environments.

This study aims to evaluate and compare the performance of multiple single-board computers in executing moderately complex image recognition tasks, more specifically, object recognition, using neural network algorithms.We chose higher-end SBC devices such as the NVIDIA Jetson Xavier, Raspberry Pi 4, Raspberry Pi 2, and Radxa Rock 5B for testing, as they are compact enough to be integrated into applications like autonomous vehicles or F1tenth cars [Zhang and Loidl, 2023]. To evaluate their performance, we first utilized the traditional Canny edge detection algorithm as a straightforward benchmark, followed by the more complex, data-driven YOLO v4 neural network algorithm. Experiments were conducted using custom-written test scripts. These devices were assessed on their ability to manage image files of various sizes.

Our results highlight a clear performance difference between the SBCs, with the NVIDIA Jetson Xavier and Radxa Rock 5B showing excellent performance in most test scenarios, especially in terms of processing time and memory efficiency.Furthermore, our tests confirmed that all selected devices are capable of effectively running complex neural network programs such as YOLO v4. This demonstrates their suitability for demanding image recognition tasks, highlighting the robust computational power and versatility of these devices.

This study provides empirical evidence for selecting the most suitable SBCs for efficient image recognition tasks. Future work will explore more algorithm optimizations and hardware configurations to further improve the efficiency and accuracy of image processing tasks.

Table of Contents

Chapter 1

Introduction

1.1 Aim and Objectives

1.1.1 Aim

This study aims to evaluate and compare the performance of multiple single-board computers (SBCs) in executing moderately complex image recognition tasks. As a starting point, we first employ the traditional Canny edge detection algorithm, which is a basic object recognition algorithm. Subsequently, to explore more challenging applications, this research introduces the advanced neural network algorithm—YOLO v4—to determine the most performant SBC in more complex image recognition tasks. Through this approach, we are able to provide a comprehensive performance comparison and thoroughly assess the capabilities of each SBC.

1.1.2 Objectives

To achieve the above aim, this study has set the following specific objectives:

1. **Performance Evaluation Setup and Experimental Design**: Select multiple SBCs for testing, including NVIDIA Jetson Xavier, Raspberry Pi 4, Raspberry Pi 2, and Radxa Rock 5B, and design experiments based on the YOLO v4 and Canny edge detection algorithms to evaluate and compare their performance in image recognition tasks.

2. **Detailed Analysis of Memory and Time Consumption**: Use the Valgrind tool to record and analyze the memory usage of each SBC when executing the YOLO v4 and Canny edge detection algorithms on small, medium, and large images. Memory use efficiency and processing speed are the main performance evaluation metrics, allowing us to assess how each SBC handles different scales of input data effectively.

3. **Consideration of Diverse Input**: In this study, to comprehensively evaluate the performance of single-board computers (SBCs) in processing data of varying

scales, we carefully selected images of three different sizes as experimental inputs. We chose two datasets, `VOCtrainval_2012` [Redmon, a]and `DIV2K_train_HR` [Computer Vision Lab ETH Zurich,], based on the following considerations:

`VOCtrainval_2012`**:** This dataset is known for its diversity and complexity of content, including a wide range of object categories and scenes, making it suitable for assessing the performance of image recognition algorithms. The average image size is about 200KB, representing smaller image samples, which are used to test the efficiency and accuracy of SBCs in processing lower-resolution images. Images from this dataset were selected as small size image test samples.

`DIV2K_train_HR`**:** Contains high-resolution images, with large images averaging 6MB and medium images averaging 2MB, to test the ability of SBCs to handle high-resolution images. This dataset allows us to assess the performance of each SBC in handling more complex and data-intensive tasks. Images from this dataset were selected as medium and large size image test samples.

Through this diverse input of images, this study ensures the comprehensiveness and reliability of the experimental results, providing a detailed reflection of the performance differences of SBCs in various types of image processing tasks.

4. **Comprehensive Performance Analysis**: Based on experimental data, comprehensively analyze the performance of each SBC in executing image recognition tasks, including metrics such as memory usage and processing time, to identify the most performance-efficient SBC.

5. **Exploration of Performance Influencing Factors**: Analyze internal factors such as CPU and clock rate, as well as external factors like temperature and fans, to deeply understand the key factors affecting performance.

1.2 Project Evaluation

This research conducts a systems evaluation, measuring the performance and behavior of software used in image recognition tasks without involving human testers or collecting any private data. We evaluate and compare the performance of multiple single-board computers (SBCs) in executing complex image recognition tasks to determine which hardware platform exhibits the best performance under specific algorithms. By integrating advanced neural network algorithms like YOLO v4 with traditional methods such as Canny edge detection, this project not only explores the capabilities of SBCs in high-level image processing tasks but also highlights the performance differences between various hardware setups and algorithms. This approach ensures that all activities are conducted with strict adherence to ethical standards in research.

1.2.1 Innovation

The innovation of this study lies in its comprehensive use of modern neural network technology and traditional image processing methods to system evaluate the performance of SBCs. This approach not only offers a new perspective for evaluating SBCs but also provides empirical foundations for hardware selection in the image processing domain. Moreover, by considering images of different sizes as inputs, this research further deepens the understanding of hardware performance under various workloads.

1.2.2 Practical Application Value

With the proliferation of artificial intelligence and machine learning applications, the demand for hardware capable of efficiently executing complex algorithms is increasing. The results of this study help guide engineers and researchers in selecting the most suitable SBC platforms for designing and implementing image recognition and related AI applications, optimizing the performance-cost ratio.

1.2.3 Contribution to Existing Research

By providing a comprehensive performance evaluation of multiple SBCs, this study fills a gap in the existing literature, offering valuable data and insights for subsequent research. Especially in terms of performance comparison between advanced algorithms like YOLO v4 and traditional algorithms such as Canny edge detection, this research provides new understandings and explanations.

1.2.4 Challenges and Limitations

Despite significant achievements in hardware performance evaluation, this study faces some challenges and limitations. For instance, the performance of hardware is influenced not only by internal configurations but may also be affected by external environmental factors such as temperature changes. Moreover, due to resource and time constraints, this research is limited to a few SBCs and specific algorithms; future studies could expand to more hardware platforms and algorithms for more comprehensive insights.

Chapter 2

Literature Review

2.1 Introduction

With the rapid development of artificial intelligence and the Internet of Things (IoT) technologies, single-board computers (SBCs) play an increasingly important role in various application fields. As compact, cost-effective, and powerful computing platforms, SBCs not only offer endless possibilities for enthusiasts and educators but also demonstrate great application potential in professional fields such as industry[Jovanović et al., 2014], healthcare, and autonomous driving[Zhang and Loidl, 2023]. Particularly in image processing[Manore et al. and computer vision tasks, SBCs, due to their portability and flexibility, have become a popular choice in research and development.

This literature review aims to explore the current state and trends of SBC applications in image processing tasks, with a special focus on hardware progress, the latest achievements in image processing and machine learning algorithms, and the challenges and solutions when combining the two. Through this review, we hope to provide a comprehensive overview of SBC research and applications in the field of image processing, while identifying gaps in current research and potential future directions.

Before diving deeper into the discussion, it is necessary to define some key terms and concepts:

- **Single-board Computer (SBC)**: A small computing device that integrates a processor, memory, input/output (I/O) functionalities, and other necessary features on a single circuit board. Typical examples include the Raspberry Pi, Arduino, and NVIDIA Jetson series.
- **Image Processing**: The process of using algorithms to analyze and modify images, aimed at improving image quality or extracting useful information. Tasks in image processing may include image enhancement, filtering, edge detection, image classification, etc.
- **Computer Vision**: A field of study that investigates how computers can "see" and "understand" the world through images and videos. It involves complex tasks

such as object recognition, motion tracking, and scene reconstruction.

- **Machine Learning and Neural Networks**: Machine learning is a technique that allows computers to learn from data and make decisions or predictions. Neural networks, a special type of machine learning model inspired by the structure and function of the human brain, are particularly good at handling tasks related to pattern recognition and classification.

By examining the latest advancements and challenges in these areas, we can better understand the potential applications and limitations of SBCs in image processing and computer vision tasks.

2.2 Theoretical Background

2.2.1 Definition and Historical Context of Single-Board Computers (SBC)

Single-board computers (SBCs)[Wikipedia, d] are complete computer systems with all essential components integrated on a single circuit board, including the processor (CPU), memory (RAM), input/output (I/O) interfaces, and necessary communication interfaces such as USB ports and network connections. The design of SBCs aims to offer a compact, cost-efficient, and fully functional solution suitable for a wide range of applications, from educational and hobbyist projects to complex industrial and scientific applications.

The history of SBCs can be traced back to the 1970s when they were primarily used for educational purposes and computer science research. With technological advancements and cost reductions, SBCs have become increasingly popular. In the early 2000s, with the introduction of devices like Raspberry Pi and Arduino, SBCs began to enter the mainstream market, expanding their application range from basic learning tools to IoT, smart home, and edge computing, among many other fields.

2.2.2 The Role of CPUs in SBCs

The Central Processing Unit (CPU) [Wikipedia, b] the brain of the SBC, responsible for executing program instructions and processing data. Its performance directly affects the overall speed and efficiency of the system. Key functions of the CPU include instruction decoding, arithmetic logic operations, and control signal output. In SBCs, the performance of the CPU is crucial as it must efficiently handle tasks within limited physical space and power supply, especially during image processing and complex computational tasks.

With technological evolution, SBC CPUs have transitioned from single-core to multi-core, integrating more advanced features such as GPU acceleration and Neural Processing Units (NPUs) to meet growing computational demands.

2.2.3 Basic Principles of Image Processing

Image processing involves techniques to achieve desired results by applying various algorithms to images as inputs. Its primary tasks include image enhancement, filtering, edge detection, image classification, and image reconstruction, among others[Rani, 2017]. Image processing has a wide range of applications, from digital photography and medical imaging to satellite image analysis and autonomous driving, covering nearly all fields requiring image analysis[Rai, 2018].

Image processing is closely related to computer vision, but they differ in focus. Image processing mainly concerns the improvement of the image itself and the extraction of information, while computer vision aims to understand the scene and objects within an image, simulating the visual perception capability of human eyes. Image processing is often a preliminary step in computer vision systems, providing preprocessed image data for subsequent advanced analysis and decision-making[Andrews, 1976].

By combining advanced image processing algorithms with the portability and accessibility of SBCs, researchers and developers can achieve efficient, real-time image analysis in various environments and applications. As algorithms and hardware technology continue to advance, the application of SBCs in the field of image processing is expected to expand and deepen further.

2.2.4 Differences Between Single-Board Computers and Traditional Computers

Single-board computers (SBCs) and traditional computers differ in several key aspects, primarily reflected in their design philosophy, performance, cost, and application range.

1. **Design Philosophy**: The design philosophy of SBCs emphasizes integration and compactness, aiming to incorporate all necessary computing functions into a single circuit board to support specific applications and development needs. In contrast, traditional computers, especially desktops and laptops, are designed for general use, with components (such as CPU, memory, storage, etc.) that can typically be independently replaced and upgraded.

2. **Performance**: Traditional computers usually offer higher computing performance, with faster CPUs, more memory, and larger storage space. In comparison, SBCs might have performance constraints but are sufficient for specific application requirements, such as IoT devices, smart home systems, and basic image processing tasks.

3. **Cost**: SBCs are relatively low-cost, making them an ideal choice for educational projects, hobbyist projects, and large-scale deployment applications. The cost of traditional computers is generally higher, considering hardware, software, and maintenance.

4. **Application Range**: Due to their low cost, small size, and flexibility, SBCs are particularly suited for embedded systems, education, prototype design, and specific

industrial applications. Traditional computers are more suited for applications requiring higher computing power, such as advanced graphics processing, gaming, and complex data analysis.

2.2.5 Introduction to Edge Detection Algorithms

Edge detection is a fundamental task in image processing, aimed at identifying the locations of edges in an image to highlight the structural features of the image. Edge detection algorithms identify edge locations by detecting abrupt changes in pixel brightness, which often represent object contours, surface texture changes, or other visual feature changes.

The Canny edge detection algorithm is a widely used edge detection technique, known for its excellent detection performance and low error rate[Han et al., 2013]. The main steps of the Canny algorithm include:

1. **Noise Reduction**: Use a Gaussian filter to smooth the image, reducing the impact of noise on edge detection.
2. **Gradient Calculation**: Use operators, such as the Sobel operator, to calculate the gradient magnitude and direction for each pixel, identifying potential edges.
3. **Non-maximum Suppression**: Check pixels along the gradient direction, retaining only local maximum values to refine edges.
4. **Double Thresholding**: Determine strong and weak edges by setting high and low thresholds, then use edge linking techniques to connect weak edges to strong edges, determining the final edges.

The Canny edge detection algorithm is favored for providing clear, continuous edges and is widely applied in various image processing scenarios, such as object recognition, scene reconstruction, and image segmentation.

CPU Clock Frequency

CPU clock frequency, often referred to as clock speed, is the rate at which a Central Processing Unit (CPU) executes its operations. This frequency is measured in Hertz (Hz), with modern computer CPUs typically operating in Gigahertz (GHz), which equates to a billion cycles per second[Wikipedia, c].

Meaning of Clock Frequency

Clock frequency is the rate of the internal clock cycles within a CPU, determining the number of clock cycles the CPU can perform per second. During each clock cycle, the CPU can execute part of a basic operation, such as reading instructions, performing calculations, or accessing memory. The higher the CPU's clock frequency, the more operations it can perform per second.

Relationship Between Clock Frequency and Performance Speed

Clock frequency is one of the key factors affecting CPU performance[Choi et al., 2014], directly relating to the speed at which the CPU operates. A high-frequency CPU can theoretically complete tasks faster because it can process more operations per second. However, the actual performance of a CPU is also influenced by other factors, such as:

1. **Number of Cores**: A multi-core CPU can handle multiple tasks or different parts of a program simultaneously, thereby enhancing overall performance.
2. **Instruction Set**: The instruction set supported by the CPU design and its efficiency can also significantly impact performance.
3. **Cache Size**: A larger cache can store more data, reducing the frequency of CPU accesses to main memory, thus speeding up processing.[kuen Kim and Song, 2017]
4. **Memory Speed**: A fast memory system that matches the CPU speed can improve overall computing speed.
5. **Data Path Width**: The width of the CPU's data paths determines the amount of data that can be transferred in each clock cycle.

Therefore, while the clock frequency of a CPU is an important metric, it does not solely determine a computer's overall performance. For example, a CPU with a lower clock frequency but more cores and a more optimized architecture may perform better in multitasking and complex applications than a high-frequency single-core CPU.

In summary, the clock frequency of a CPU is one of the key factors in measuring its operational speed, but assessing the overall performance of a CPU requires consideration of several other hardware and design parameters.

2.3 Trends in Single-Board Computer Development

In recent years, technological advancements in single-board computers (SBCs) have not only enabled their transition from simple educational tools to complex industrial and research projects but have also significantly enhanced their processing capabilities to meet the demands for more intensive tasks. Notably, products like the NVIDIA Jetson series and Raspberry Pi 4 have demonstrated powerful performance and broad application potential in edge computing, deep learning, and education.

2.3.1 Technological Advancements

The notable progress in SBC hardware technology, especially in areas such as CPU, GPU acceleration, and memory management, has been remarkable. For instance, the NVIDIA Jetson series has greatly improved processing capabilities for edge computing and deep learning tasks by integrating a high-performance Volta GPU with a multi-core Carmel ARM CPU. Additionally, the introduction of Raspberry Pi 4, offering higher

processing speeds and greater memory options, has provided enhanced performance and flexibility for educational and hobbyist projects.

2.3.2 Emerging Products

The market has seen the emergence of new SBC products optimized for image processing and machine learning tasks, showcasing the rapid development of SBC technology. Products like NVIDIA Jetson Nano and Jetson Xavier NX, with powerful GPU acceleration and efficient energy management, have opened new possibilities for edge computing and AI inference. The release of these products reflects the growing demand for high-performance SBCs in research, industrial, and commercial applications.

2.3.3 Application Scenarios

The application scenarios of SBCs have expanded from initial educational tools to encompass home automation, industrial control, and research projects, among others. For example, research by Kim and Zabik (1992) developed a single-board computer system for scientific instruments[Kim and Zabik, 1992], showcasing the potential of SBCs in scientific research. Furthermore, Ariza and Báez (2021)[Ariza and Báez, 2021] conducted a systematic literature review on the application of single-board computers in education[Ariza and Baez, 2021], highlighting their importance in laboratories, computing education, and the Internet of Things (IoT).

2.3.4 Contribution of the NVIDIA Jetson Series

The NVIDIA Jetson series' contribution to edge computing and deep learning primarily lies in its powerful GPU acceleration capabilities and efficient deep learning model training and inference performance. Research by S.K Prashanthi et al. (2022)[S.K et al., 2022] analyzed the performance of deep neural network (DNN) model training on Jetson devices, offering important insights into tuning performance for training on the edge. Additionally, Holly et al. (2020)[Holly et al., 2020] conducted an in-depth power consumption analysis of the NVIDIA Jetson Nano, providing valuable guidance for achieving low power consumption and efficient energy management on edge devices.

2.3.5 Conclusion

The advancements in hardware technology, the introduction of new products, and the diversified application scenarios have underscored the growing potential and importance of single-board computers. From education to industrial control, from research to deep learning, SBCs are emerging as a pivotal tool for driving technological innovation and accomplishing complex task processing. With ongoing technological progress and product innovation, SBCs are poised to play an even greater role in image processing and machine learning, paving new paths for future technological developments.

2.4 Trends in Image Processing

In the field of image processing technology, the rapid development of deep learning and computer vision algorithms is significantly changing the approach to tasks such as image recognition, classification, and analysis. Particularly, algorithms based on deep learning like Convolutional Neural Networks (CNNs) and the YOLO (You Only Look Once) series have become a transformative force in this field due to their superior performance. These advancements offer notable advantages in terms of accuracy and processing speed over traditional methods but also require greater computational resources. Therefore, a key challenge lies in optimizing these algorithms to fit within the computational constraints of SBCs.

2.4.1 Algorithm Innovation

Significant advancements in image processing and computer vision are largely driven by the application of algorithms such as CNNs and the YOLO series. The study by Moen et al. (2019) [Moen et al., 2019]highlighted the use of deep learning algorithms in biological image analysis, altering the methodology of analyzing and interpreting image data, thereby making previously challenging datasets more manageable and opening up new possibilities for experimentation. Additionally, Hao and Hao (2020) [Hao and Hao, 2020]reviewed the progress in deep learning-based Optical Coherence Tomography (OCT) image processing techniques, underscoring achievements in aspects like image recognition, segmentation, enhancement, and denoising.

2.4.2 Performance Improvement

With the optimization of algorithms, application of parallel processing techniques, and utilization of hardware acceleration, significant improvements have been made to the efficiency and accuracy of image processing tasks. Optimizing deep learning algorithms for execution on resource-constrained devices like SBCs has enhanced the performance of edge devices in complex image analysis tasks. Chen et al. (2021) [Crusio et al., 2020]reviewed recent advancements in the application of deep learning methods for medical image processing tasks, highlighting achievements in unsupervised and semi-supervised learning, and demonstrating the potential of deep learning to improve image processing accuracy.

2.4.3 Software Tools

The support for SBCs by popular image processing and machine learning frameworks such as TensorFlow and PyTorch is on the rise. Jayaraman et al. (2023)[Jayaraman et al., 2023] conducted a comparative study of TensorFlow, Pytorch, Caffe, and Keras in a single-image deraining application, finding TensorFlow and PyTorch to offer superior results in terms of prediction accuracy, CPU optimization, convergence factors, and memory usage. Furthermore, Alaeddine et al. (2021) [Alaeddine et al., 2021]detailed the design and implementation of a hardware/software accelerator integrated into the TensorFlow framework, enabling efficient execution of deep learning models on FPGAs.

These advancements not only showcase the application potential of deep learning in image processing but also emphasize the importance of enhancing performance and optimizing resources on edge devices like SBCs. Future research will focus on further optimizing algorithms and frameworks under limited resource conditions, aiming to improve the efficiency and accuracy of SBCs in performing image processing tasks.

2.5 Applications of Single-Board Computers in Image Processing

Single-board computers (SBCs) have become extensively used in the field of image processing, playing a crucial role in a wide range of tasks from edge detection to complex object recognition and tracking. Devices such as the Raspberry Pi and NVIDIA Jetson series support the development of home surveillance systems and real-time face recognition systems and are vital in demanding applications like drones, autonomous vehicles, and industrial automation.

2.5.1 Models and Types

In the realm of image processing research, the Raspberry Pi and NVIDIA Jetson series are commonly utilized models of SBCs. The Raspberry Pi is lauded for its cost-effectiveness and strong community support, making it ideal for educational and hobbyist projects. Meanwhile, the NVIDIA Jetson series, with its high-performance GPU acceleration capabilities, is particularly suited for edge computing and deep learning applications, offering significant performance advantages. An example being the Jetson TK1 platform, which, with its on-board computing capabilities, is ideal for real-time image processing applications due to reduced communication overhead and latency.[Pundlik et al., 2017]

2.5.2 Task Categories

SBCs handle a vast array of image processing tasks, including object detection, facial recognition, and image enhancement. Integrating camera modules with the Raspberry Pi facilitates the building of home surveillance systems, while the NVIDIA Jetson can develop advanced systems capable of real-time facial recognition in dynamic environments. Additionally, SBCs find widespread use in drone image processing, autonomous vehicle vision systems, and industrial automation visual inspection, where the demands for real-time performance and accuracy are exceptionally high.[Yildirim et al., 2022]

2.5.3 Performance Evaluation

Evaluating performance is crucial in assessing the application effectiveness of SBCs in image processing. Comparison of performance metrics across studies, such as processing speed, accuracy, and energy efficiency, provides insights into the practical performance of SBCs for specific image processing tasks. For example, Wagh [Shilpashree et al., 2015]demonstrated the applicability of the OpenCV library on an SBC, showcasing the utility of the Qt and

Linux gcc Integrated Development Environment (IDE) for image processing algorithms. Furthermore, Prasad and Sinha [Prasad and Sinha, 2011]showcased the capability of SBCs in complex real-time scenarios with the implementation of an object detection and tracking system through real-time video processing using a single camera.

In summary, with their compact size, powerful performance, and broad application potential, SBCs are playing an increasingly important role in the field of image processing. Continuous technological innovation and algorithm optimization are expected to further expand the application spectrum of SBCs in image processing and computer vision, offering practical guidance for researchers and developers in selecting hardware and designing algorithms.

2.6 Discussion on YOLO v4 and Canny Edge Detection Algorithm

2.6.1 Introduction to YOLO v4

YOLO (You Only Look Once) v4 is a state-of-the-art real-time object detection system introduced by Alexey Bochkovskiy in 2020, designed to balance high detection accuracy with high processing speed, suitable for real-time applications. It predicts object classes and locations from images in a single forward pass, making it highly efficient.

Implementation Mechanism YOLO v4 employs a convolutional neural network (CNN) architecture, dividing the input image into an S×S grid. Each grid cell predicts multiple bounding boxes and their corresponding class probabilities, incorporating several innovative features in its architecture:

- **CSPDarknet53**: Serves as the backbone network for feature extraction.
- **Spatial Pyramid Pooling (SPP)**: Enhances the network's receptive field and scale adaptability.
- **Path Aggregation Network (PAN)**: Improves feature integration by optimizing the feature transfer mechanism.
- **Anchor Boxes**: Uses predefined bounding box dimensions, determined through cluster analysis, to improve the prediction accuracy.

Mathematical Expression Let $P(class|object)$ represent the probability of an object belonging to a certain class given its presence, $P(object)$ the probability of object presence within a grid cell, and b_w, b_h, b_x, and b_y denote the dimensions and center coordinates of the bounding box, respectively.

2.6.2 Introduction to Canny Edge Detection

Introduced by John Canny in 1986, the Canny edge detection algorithm is a multi-stage technique to extract high-quality edges from images, known for its excellent performance and low error rates.

Implementation Mechanism The Canny algorithm identifies edges by detecting abrupt changes in pixel brightness, following several key steps:

1. **Noise Reduction**: Application of a Gaussian filter to smooth the image.
2. **Gradient Calculation**: Use of Sobel operators to calculate gradient magnitude and direction for each pixel.
3. **Non-maximum Suppression**: Retention of local maxima along the gradient direction to refine edges.
4. **Double Thresholding and Edge Tracking**: Use of high and low thresholds to identify true and potential edges, followed by edge linking to form complete edges.

2.6.3 Comparison of YOLO v4 and Canny Edge Detection

Features, Advantages, and Limitations - **YOLO v4** excels in fast, accurate real-time object detection, suitable for rapid response scenarios. However, it may show lower accuracy for small objects and demands higher computational resources. - **Canny Edge Detection** is highly accurate in edge extraction and noise suppression, ideal for applications needing precise edge information. Its limitations include difficulties in processing complex scenes and achieving real-time performance.

Applicability - **YOLO v4** is ideal for real-time video surveillance, autonomous vehicle vision systems, and drone vision recognition. - **Canny Edge Detection** is best suited for tasks requiring detailed edge information, such as image pre-processing and feature extraction in medical imaging.

Impact of Algorithm Selection The choice of algorithm significantly influences the performance of image processing tasks on SBCs. Efficient algorithms enable fast and accurate processing within the computational limits of SBCs, while complex algorithms may introduce delays, impacting real-time performance. Thus, the selection should be based on the specific application requirements and the computational capabilities of the SBC.

2.7 Methods of Performance Evaluation

To accurately assess the performance of single-board computers (SBCs) in executing image processing tasks, this study employs a comprehensive performance evaluation

framework. This approach is designed to understand thoroughly and compare the performance characteristics across various hardware platforms. The following outlines the evaluation metrics and methods used for comparison.

2.7.1 Evaluation Metrics

For an objective evaluation of SBCs' performance in image processing tasks, we have identified key metrics:

1. **Processing Time**: This refers to the total time required from inputting an image to obtaining the processed output. It includes durations needed for image reading, preprocessing, algorithm execution, and result generation. Processing time is crucial for assessing the real-time capabilities of a system, particularly vital for applications demanding swift responses, such as autonomous driving and real-time surveillance.

2. **Memory Usage**: The peak memory consumption during the execution of the algorithm, reflecting the computational resource demands of the algorithm. On SBCs, where resources are limited, efficient memory utilization is key to ensuring system stability and scalability.

2.7.2 Comparison Methods

The study adopts a systematic approach to evaluate and compare the performance of different SBCs, as described below:

1. **Experimental Design**: A uniform testing environment is established to ensure the fairness and comparability of the evaluation results across all devices. The experiments involve both advanced deep learning algorithms, like YOLO v4, and traditional algorithms such as Canny edge detection, to encompass a broad spectrum of image processing needs.

2. **Selection of Datasets**: Diverse and representative image datasets, such as VOC-trainval_2012 and DIV2K_train_HR, are chosen to facilitate a comprehensive evaluation. These datasets include images of varying resolutions and complexities, enabling an extensive assessment of algorithm performance under different conditions.

3. **Measurement of Performance Metrics**: Specialized tools, including Valgrind for analyzing memory usage and built-in timing functions for measuring processing times, are utilized to ensure the accuracy and reliability of the collected data.

4. **Analysis and Comparison of Results**: The performance of each SBC is meticulously analyzed and compared based on the specified metrics. This analysis helps identify the strengths and limitations of each hardware platform and provides insights into the efficiency of algorithms and their adaptability to specific hardware configurations.

Through the application of this methodical performance evaluation approach, the study aims to offer empirical evidence supporting the selection of the most suitable SBCs for efficient image processing tasks. Additionally, it emphasizes the importance of algorithm optimization in improving image processing performance. By taking into account metrics such as processing time, memory usage, and accuracy, a comprehensive evaluation of the image processing capabilities of various SBCs is achieved. This evaluation provides valuable insights for researchers and developers regarding hardware selection and algorithmic adjustments.

2.8 Discussion

This literature review has conducted an in-depth analysis and comparison of various studies on the performance of single-board computers (SBCs) in image processing tasks, with the goal of understanding the potential and challenges of SBCs in the current technological environment. Through comparison, it becomes evident that despite differences in research focus and application scenarios, common findings and challenges emerge across studies.

2.8.1 Comparative Analysis of Research Findings

The majority of studies highlight that with the advancement of deep learning and computer vision technologies, SBCs are increasingly utilized across a broad spectrum of image processing applications. These range from basic tasks like image classification and edge detection to more complex functions such as real-time object detection and tracking. Despite the limitations in computational power and resources, SBCs can efficiently execute advanced image processing tasks through algorithm optimization and hardware improvements, including the application of YOLO v4.

Differences among studies are also noteworthy. For instance, some research prioritizes the enhancement of algorithm efficiency and accuracy[Holly et al., 2020], while others focus on optimizing hardware performance and power management[S.K et al., 2022]. The selection of algorithms and testing datasets further contributes to the diversity in performance evaluation outcomes.

2.8.2 Contributions to Theory and Methodology

Existing literature makes significant contributions to both the theoretical framework and methodological approaches in the domain of image processing using SBCs. The exploration of SBCs' capabilities in running deep learning algorithms such as YOLO v4 not only demonstrates their potential in managing complex image recognition tasks but also encourages further research into algorithm optimization and hardware adaptation. The introduction of YOLO v4, in particular, has extended the frontiers of object detection technology, providing valuable insights into executing sophisticated algorithms on devices with limited resources.

Furthermore, these studies offer crucial insights for future scientific exploration and technological innovation, emphasizing the importance of algorithm optimization to enhance processing speeds and reduce resource consumption, as well as the need for hardware improvements to boost computational power and energy efficiency.

2.8.3 Directions for Future Research

From the synthesis of existing research, several avenues for future investigation are identified:

1. **Cross-platform Algorithm Generality Studies**: Investigating ways to enhance algorithm generality, enabling efficient execution across various SBCs without necessitating specific hardware adaptations.
2. **Co-Optimization of Algorithms and Hardware**: Delving deeper into the interplay between algorithms and hardware to explore co-design strategies that maximize performance and energy efficiency.
3. **Development of Low-Resource Consumption Deep Learning Models**: Creating innovative deep learning models and training methods that lower resource usage, making them more apt for deployment on SBCs and other devices with constrained resources.

Addressing these gaps in research could propel the use of SBCs in image processing and computer vision forward, opening up new possibilities for innovation and wider application fields.

2.9 Research Gaps and Future Directions

Despite significant progress in existing research, many research gaps and critical issues in image processing applications on single board computers (SBCs) remain unexplored. Addressing and exploring these issues is crucial for setting future research directions.

Limitations of Existing Research

1. **Algorithm and Hardware Compatibility**: Most research focuses on the performance evaluation of specific algorithms on specific hardware, with less consideration given to the deep compatibility and collaborative optimization between algorithms and hardware. This results in potentially significant performance variations when the same algorithms are run on different SBCs.
2. **Stability and Reliability in Dynamic Environments**: Current research largely evaluates performance in static or controlled environments, with insufficient research on the stability and reliability of algorithms in dynamic environments.

3. **Optimization of Energy Efficiency Ratio**: While some studies have considered the energy efficiency of algorithms, how to optimize the energy efficiency ratio of SBCs while maintaining algorithm performance remains an important and underexplored area.

4. **Real-time Processing Capabilities**: For applications requiring real-time processing, such as autonomous driving and real-time monitoring, current research is insufficient in enhancing the processing speed of SBCs and reducing latency.

5. **Scalability for Large-scale Deployment**: Existing research often focuses on performance evaluation on a single device, with insufficient discussion on performance maintenance and resource management during large-scale deployment.

Future Research Suggestions

Algorithm-Hardware Co-design

Future research should explore more on the co-design of algorithms and hardware. By customizing algorithms to fully exploit the performance characteristics of specific hardware platforms, or by adapting hardware to suit specific image processing algorithms. Marek Wnuk's (2008) study discussed implementing specific features of image processing algorithms on hardware, emphasizing the importance of hardware optimization for real-time image processing[Wnuk, 2008]

Algorithm Research for Dynamic Environments

Develop and optimize image processing algorithms that can adapt to changes in dynamic environments, improving stability and reliability in non-ideal conditions. For instance, He and Xiong's (2021) study on image processing design and algorithms under cloud computing technology demonstrates the possibilities of processing image data in a broader computing environment[He and Xiong, 2021].

Optimization Techniques for Energy Efficiency Ratio

Investigate new techniques for optimizing the energy efficiency ratio, including algorithmic optimizations and hardware-level energy management strategies, to meet the needs of energy-limited scenarios. James Hegarty and others (2016) proposed a new multirate architecture that can optimize image processing algorithms at the hardware level, further improving processing speed and energy efficiency[Hegarty et al., 2016].

Enhancing Real-time Processing Capabilities

Explore new data processing frameworks and algorithm optimization methods to reduce image processing latency and enhance the real-time processing capabilities of SBCs. Yu and Leeser (2005) discussed a streamlined design process for windowed image processing applications that can fully utilize on-board resources to enhance performance[Yu and Leeser, 2005].

Scalability and Large-scale Deployment

Research image processing solutions aimed at large-scale deployment, including distributed processing architectures and resource management strategies, to support efficient operation in a multi-device collaborative environment.

These research directions not only aim to advance the application of single board computers in the field of image processing but also open up broader innovative spaces and prospects for related technologies. Especially in the area of algorithm and hardware co-design, it is expected to unlock new chapters of SBC performance potential, bringing more possibilities to edge computing.

2.9.1 Conclusion

The investigation of single-board computers in the realm of image processing has unveiled vast possibilities in fields ranging from autonomous systems to environmental monitoring. With technological progress, the potential applications of SBCs continue to grow. Future research directed towards addressing the present gaps and venturing into the proposed directions promises to further augment the functionalities of SBCs, rendering them even more potent and accessible tools for technological innovation.

2.10 Implementing Future Research Directions and Broader Impacts

The advancements in single-board computers (SBCs) for image processing highlight significant opportunities for technological innovation, presenting challenges that necessitate collaborative efforts across various disciplines for effective implementation of the proposed future research directions.

2.10.1 Implementing Research Directions

1. **Interdisciplinary Collaboration**: Addressing the research gaps outlined requires a collaborative approach that spans computer science, electrical engineering, user experience design, and domain-specific expertise. For example, efforts to optimize algorithms for energy efficiency will benefit from the combined knowledge in hardware design and user-centric software development.

2. **Open Source and Community Engagement**: Utilizing open-source projects and fostering community-driven development can significantly accelerate advancements in making SBCs more accessible and robust for image processing tasks. Collaboration platforms like GitHub serve as a hub for sharing, collaboration, and iteration on both software and hardware design enhancements.

3. **Industry Partnerships**: Collaboration with industry partners can offer valuable insights into the practical applications and challenges of SBCs in image processing, steering research towards outcomes that are both commercially viable and socially

beneficial. Such partnerships can also provide essential resources for the testing and deployment of new innovations.

2.10.2 Broader Impacts

The implications of enhancing SBC technology for image processing reach far beyond the technical realm:

1. **Educational Advancements**: Improvements in SBC technology can revolutionize education, offering affordable and tangible tools for hands-on learning in STEM areas, especially in robotics, computer vision, and environmental monitoring.
2. **Societal Benefits**: Enhanced image processing capabilities on SBCs can contribute to societal well-being in critical areas such as healthcare diagnostics, environmental conservation, and security systems through advanced real-time analysis and object detection.
3. **Technological Innovation**: The increased capabilities and accessibility of SBCs serve as a catalyst for innovation, empowering startups and hobbyists to rapidly and cost-effectively bring new ideas to market.

2.10.3 Conclusion

The exploration and enhancement of single-board computers in the domain of image processing are far from complete. By addressing the gaps identified and pursuing the suggested research directions, the potential of SBCs to impact technology and society positively is immense. The collaborative efforts across disciplines, augmented by community engagement and industry partnerships, will be crucial in unlocking the full potential of SBCs, setting the stage for a revolution in image processing and beyond.

Chapter 3

Requirements Analysis

3.1 Criteria for Selecting Experimental Equipment

Prior to the commencement of this experiment, the following single-board computers were shortlisted as candidates3.1.

In this study, four single-board computers were meticulously selected: NVIDIA Jetson Xavier NX, Raspberry Pi 4, Raspberry Pi 2B, and Radxa Rock 5B, to evaluate and compare their performance in executing complex image recognition tasks. The rationale behind choosing these specific devices is grounded on the following key considerations:

1. **Experimental Design for Control**: The selection of Raspberry Pi 2B and Raspberry Pi 4 aimed at facilitating controlled experiments. These two devices exhibit significant differences in performance and release times, which can help in observing and analyzing the progression of single-board computer performance over time.

2. **Popularity and Widespread Use**: Raspberry Pi 4B, NVIDIA Jetson Xavier NX, and Radxa Rock 5B, as the more popular, hot, and widely used single-board computers in recent years, ensure that the results of this study are highly relevant and valuable. Their extensive range of applications also implies that their performance in handling different types of tasks warrants in-depth exploration.

3. **Architectural Consistency**: All selected single-board computers are based on the ARM architecture. The ARM architecture, being the mainstream choice in the field of single-board computers, offers lower power consumption and higher energy efficiency ratios. Choosing devices based on the ARM architecture ensures the wide applicability and reference value of the experimental results. Although single-board computers based on the x86 architecture appeared in the candidate list, they were not included in this experiment due to their lesser relevance in the single-board computer market.

4. **Performance Variation**: Viewing from the CPU's clock speed, these four single-board computers demonstrate different processing speeds, expected to exhibit sig-

Feature/Board	CPU	Clock Speed	GPU	RAM
Raspberry Pi 4 Model B	4-core Cortex-A72 architecture	1.5GHz	Broadcom VideoCore VI	1GB, 2GB, 4GB or 8GB LPDDR4-3200 SDRAM (depending on model)
Nvidia Jetson Xavier NX	6-core NVIDIA Carmel ARMv8.2 64-bit CPU	Up to 1.4GHz in 4/6-core mode and up to 1.9GHz in dual-core mode	384-core NVIDIA Volta GPU with 48 Tensor Cores	8GB 128-bit LPDDR4x with a memory bandwidth of 51.2GB/s
Raspberry Pi 2 Model B	Quad-core ARM Cortex-A7	900 MHz	Broadcom VideoCore IV	1GB LPDDR2
Radxa Rock Pi 5B	Rockchip RK3588 Quad-core ARM Cortex-A76 MPCore processor and quad-core ARM Cortex-A55 MPCore processor	2.4GHz for the Cortex-A76 cores and 1.8GHz for the Cortex-A55 cores	Embedded ARM Mali-G610 MP4 3D GPU	LPDDR 4x, two 32bits LPDDR 4x chips make 64bits, data frequency is up to 4224Mhz
Jetson TX2	Dual-core Denver 2 64-bit CPU and quad-core Arm Cortex-A57 MPCore processor	2.0GHz & 2.2GHz	256-core NVIDIA Pascal ™ architecture GPU	8GB 128-bit LPDDR4, with a memory bandwidth of 59.7GB/s
Jetson Nano	4-core Cortex-A57 architecture	1.43GHz	NVIDIA Maxwell, with 128 NVIDIA CUDA cores	4GB 64-bit LPDDR4
Raspberry Pi 5	Broadcom BCM2712 quad-core Arm Cortex A76 processor	2.4GHz	Broadcom VideoCore VII GPU	LPDDR4X-4267 SDRAM, available in 4GB and 8GB configurations
Radxa Rock Pi (Rock Pi 4B v1.3)	Dual CPU: Quad-core A72 + A52	A72: 1.8GHz, A52: 1.4GHz	Mali T860MP4	LPDDR4 (3200 Mbps)
Banana Pi (BPi-M3)	Octa-core ARM-A7	2.0GHz	Low-Powered GPU	2GB LPDDR3
Odroid (XU4)	Samsung Exynos 5542	1.8GHz	Not Specified	2Gbyte LPDDR3 RAM PoP stacked
Arduino MKR Zero	SAMD21 Cortex®-M0+ 32-bit low power ARM MCU	48 MHz	Not applicable	256KB Flash, 32KB SRAM
Asus Tinker Board	Quad-core ARM-basedprocessor - Rockchip RK3288	A76: 2.4GHz	ARM Mali™-T764	2GB LPDDR3 dual-channel
PINE64 Pine A64	Quad-Core Allwinner A64 @	1.152 GHz	ARM Mali 400 MP2	Up to 2GB LPDDR3
Intel NUC Boards	Intel® Core™ i3-1315U Processor	Up to 4.50GHz	Integrated in CPU	Up to 64 GB DDR4-3200 SODIMMs
LattePanda 3 Delta	Intel® Celeron® Processor N5105	2.00GHz~ 2.90GHz	Intel® UHD Graphics (450MHz~800MHz)	Samsung 8GB LPDDR4 2933MHz
LattePanda Sigma	Intel® Core™ i5-1340P	Up to 4.60GHz	Intel® Iris® Xe Graphics	Samsung LPDDR5-6400(16GB),LPDDR5-6000(32GB)

Figure 3.1: hardware list

nificant performance differences. Selecting devices with notable differences in processing speeds can help in clearly identifying and evaluating each device's capability in handling complex image recognition tasks, thereby providing a distinct comparative difference for the experimental results.

3.2 Software Selection

In the selection of suitable algorithms for image processing and object detection experiments, several algorithms were considered, including Faster R-CNN, RetinaNet, YOLO v4, YOLO v5, Canny edge detection algorithm, and SSD (Single Shot MultiBox Detector). Below is an introduction to these algorithms, along with an analysis of their advantages and disadvantages, especially when running on resource-constrained single-board computers.

3.2.1 Algorithm Principles and Analysis

YOLO v4

YOLO v4[Bochkovskiy et al., 2020][Redmon, b], an improved version in the YOLO series, maintains high processing speeds while enhancing detection accuracy. It leverages a deep convolutional neural network (CNN) architecture that directly maps image pixels to bounding box coordinates and class probabilities. This contrasts with traditional detection systems that typically propose regions before classification.

Network Structure YOLO v4 uses CSPDarknet53 as its backbone, featuring a Cross Stage Partial Network (CSPNet) that reduces computation and improves gradient flow. This speeds up training and boosts performance.

Feature Pyramid

To enhance detection across scales, YOLO v4 employs Spatial Pyramid Pooling (SPP) and Path Aggregation Network (PAN). SPP pools features at various scales, preserving spatial hierarchy information. PAN aggregates these features, enhancing their representational power.

Detection Process of YOLO v4 YOLO v4 divides the image into a $S \times S$ grid. Each grid cell predicts bounding boxes for targets whose center falls within the cell. Each bounding box comprises:

- b_x, b_y: Center coordinates relative to the grid cell.
- b_w, b_h: Width and height relative to the image dimensions.
- Confidence score: Probability that the box contains an object multiplied by the predicted IOU (Intersection over Union) with the actual object.
- Class probabilities: Probability that the object in the box belongs to each class.

Mathematical Formulas The predictive capabilities of YOLO v4 can be summarized with the following formulas:

Bounding Box Prediction

$$
\begin{aligned}
b_x &= \sigma(t_x) + c_x \\
b_y &= \sigma(t_y) + c_y \\
b_w &= p_w e^{t_w} \\
b_h &= p_h e^{t_h}
\end{aligned}
$$

where σ denotes the sigmoid function, t_x, t_y, t_w, t_h are the network's predicted adjustment parameters, c_x, c_y are the offsets from the top left corner of the grid, and p_w, p_h are preset dimensions of the bounding box.

Confidence and Class Probabilities

$$\text{confidence} = \sigma(t_{\text{conf}}) \times \text{IOU}(b, \text{object})$$

$$\text{class probability} = \sigma(t_{\text{class}})$$

YOLO v4 also incorporates numerous enhancements such as data augmentation, multi-scale training, and Batch Normalization to improve generalization and accuracy.

Canny Edge Detection Algorithm

The Canny edge detection algorithm[Wikipedia, a][DK_csdn1,] is a widely used technique in image processing for detecting edges in images. Introduced by John Canny in 1986, it is designed to offer good detection, localization, and minimal response multiplicity.

Gaussian Filtering The first step in the Canny edge detection algorithm is to apply a Gaussian filter to smooth the image to eliminate noise. Gaussian filtering is a linear filtering technique based on the Gaussian function, used to blur images. The formula for the Gaussian function is:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

where σ is the standard deviation of the Gaussian kernel, which determines the extent of filtering. During image processing, we convolve this Gaussian function with the original image.

Finding the Gradient Magnitude and Direction Edges in an image can be found at points of maximum gradient. First, we need to calculate the gradient magnitude and direction at each image pixel. The gradient magnitude assesses the strength of the edges, while the gradient direction indicates the direction of the edges.

We commonly use the Sobel operator to calculate the gradient in the horizontal direction (Gx) and the vertical direction (Gy):

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * \text{Image}$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * \text{Image}$$

The gradient magnitude and direction can then be calculated using the following formulas:

$$\text{Magnitude} = \sqrt{G_x^2 + G_y^2}$$

$$\text{Direction} = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

Non-Maximum Suppression To achieve as thin edges as possible, the algorithm performs non-maximum suppression. This step removes points that are not considered edges. Specifically, the algorithm checks if a pixel is a local maximum in its gradient direction. If it is not, the pixel value is set to zero, meaning it is not considered as an edge.

Double Thresholding and Edge Tracking The final step is to use double thresholding to determine real edges. Two thresholds are selected: a high threshold and a low threshold. Strong edges (points above the high threshold) are kept, while weak edges (points between the two thresholds) are only retained if they connect to strong edges.

This process helps eliminate false edges caused by noise and color variations, retaining more significant edge information.[tao Jin, 2010]

This concludes the detailed introduction to the Canny edge detection algorithm. Through several well-designed steps, the algorithm effectively detects edges in images, serving as a fundamental tool in the fields of computer vision and image processing.

Faster R-CNN

[Ren et al., 2016]

Core Concepts

Faster R-CNN is an object detection algorithm proposed by Ren et al. It achieves fast and efficient object detection by introducing Region Proposal Network (RPN). RPN first generates object candidate regions, then extracts features using CNN, and finally outputs the categories and positions of objects through classifiers and bounding box regressors.

Implementation Mechanism

1. **Region Proposal Network (RPN)**: RPN uses sliding windows to extract object candidate regions on the feature map. For each window position, RPN simultaneously predicts multiple region proposals and their "objectness" scores. 2. **RoI**

Pooling: Extracts fixed-size feature vectors from each proposed region on the feature map for subsequent classification and bounding box regression. 3. **Classification and Bounding Box Regression**: Uses extracted feature vectors for object classification and precise adjustment of bounding box positions.

Mathematical Expressions

The "objectness" score of RPN can be calculated using the sigmoid function, and bounding box regression is adjusted using the following formula:

$$t_x = \frac{(x - x_a)}{w_a}, \quad t_y = \frac{(y - y_a)}{h_a}, \quad t_w = \log\left(\frac{w}{w_a}\right), \quad t_h = \log\left(\frac{h}{h_a}\right)$$

where x, y, w, h represent the center coordinates and width-height of the predicted box, and x_a, y_a, w_a, h_a represent the corresponding parameters of the anchor box.

RetinaNet

Core Concepts

RetinaNet[Lin et al., 2018][Esri,], proposed by Lin et al., is an algorithm that uses Focal Loss to address class imbalance issues in object detection. It consists of a Feature Pyramid Network (FPN) and two subnetworks for object classification and bounding box regression, respectively.

Implementation Mechanism

1. **Feature Pyramid Network (FPN)**: Constructs multi-scale feature maps to adapt to detection of objects of different sizes. 2. **Focal Loss**: Modifies traditional cross-entropy loss to reduce the weights of easily classified samples, enabling the model to focus more on hard-to-classify samples. 3. **Classification and Bounding Box Regression Subnetworks**: Predicts object categories and bounding box adjustment parameters for each anchor point on the feature map.

Mathematical Expressions

The formula for Focal Loss is:

$$\text{FL}(p_t) = -\alpha_t(1 - p_t)^\gamma \log(p_t)$$

where p_t is the predicted probability of the correct class, α_t is the weight factor, and γ is the modulation factor.

SSD (Single Shot MultiBox Detector)

The SSD (Single Shot MultiBox Detector) [Liu et al., 2016][Liu,]is a popular object detection algorithm, widely acclaimed for its speed and effectiveness. This detailed explanation will cover everything from basic principles to specific mathematical formulas.

Basic Principles

The core idea of SSD is to predict the presence of multiple categories and the bounding boxes of objects in a single forward pass. This design makes SSD more efficient than traditional algorithms that require multiple passes to detect different objects.

Network Structure

The network structure of SSD is typically based on a pre-trained convolutional neural network (such as VGG16) to extract features. The subsequent part of the network includes several feature layers that detect objects at different scales, allowing SSD to detect objects of varying sizes. **Feature Layers and Predictions**

At each feature layer, SSD uses a set of default bounding boxes (also known as anchors or default boxes), which vary in scale and aspect ratio. Each default box generates a certain number of outputs through a convolutional layer, including:

- **Location Adjustments** (four values, typically represented as offsets relative to the default box)
- **Class Confidence** (a value for each class plus one for the background)

Mathematical Formulas

For location adjustments, SSD predicts the offset (cx, cy, w, h) of the predicted box $\hat{G}$ relative to the default box D, where cx, cy are the offsets for the center coordinates and w, h are the log-scale transformations of width and height:

$$\begin{aligned}
\hat{G}_{cx} &= D_w \times \text{pred}_{cx} + D_{cx} \\
\hat{G}_{cy} &= D_h \times \text{pred}_{cy} + D_{cy} \\
\hat{G}_w &= D_w \exp(\text{pred}_w) \\
\hat{G}_h &= D_h \exp(\text{pred}_h)
\end{aligned}$$

For class confidence, the probability for each class is computed using the softmax function:

$$\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

where x_i is the raw prediction score for class i.

Training

The SSD model is trained using a loss function, which is a weighted sum of the location loss (typically smooth L1 loss) and confidence loss (cross-entropy loss):

$$L = \frac{1}{N}(L_{\text{conf}} + \alpha L_{\text{loc}})$$

where N is the number of positive examples matching default boxes, L_{conf} is the confidence loss, L_{loc} is the location loss, and α is a parameter to balance the two.

This concludes the basic concepts and key mathematical formulas of the SSD algorithm. SSD effectively performs detection at multiple scales for objects of different sizes while maintaining high detection speed. This should help in understanding the workings of SSD in more detail. Feel free to ask if you have any more specific questions.

3.2.2 YOLO v4 and Canny Edge Detection Algorithm Selection Reasons

In this study, we have chosen YOLO v4 and the Canny edge detection algorithm as benchmarks for performance evaluation. The selection of these two algorithms aims to constitute a comparative experiment to comprehensively assess the performance of single-board computers in executing different types of image processing tasks. These two algorithms are chosen not only because they are both implemented in C language, ensuring consistency in the experimental environment, but also because they represent deep learning-based methods and traditional image processing methods, respectively. This contrast helps to deepen the understanding of the efficiency and applicability of different computational models on hardware with limited resources.

YOLO v4

YOLO v4, representing "You Only Look Once version 4," is an advanced real-time object detection system that can simultaneously detect multiple objects in an image, making it an exemplar of high efficiency and accuracy. The design goal of YOLO v4 is to maintain high detection accuracy while improving processing speed, making it highly suitable for applications requiring rapid response.

- **Efficiency and Accuracy**: YOLO v4 achieves high efficiency and accuracy in image recognition tasks through optimized algorithm structures and the use of more efficient pre-trained models.
- **Wide Applications and Recognition**: YOLO v4 has been widely applied and recognized in both industry and academia, especially in scenarios requiring real-time processing such as video surveillance and autonomous driving.
- **Representative of Neural Network Algorithms**: As a deep learning-based algorithm, the selection of YOLO v4 reflects the current trends in image processing and computer vision fields.

Canny Edge Detection Algorithm

The Canny edge detection algorithm is a classical image processing algorithm designed to extract useful structural information from images and reduce data volume. It identifies edges in images through mathematical computations, and it is widely used due to its efficiency and accuracy.

- **Classical and Reliable**: Since its inception, the Canny algorithm has been widely adopted for its excellent edge detection performance and low error rate in various image processing applications.

- **Role as a Comparative Benchmark**: As a representative of traditional image processing methods, the Canny algorithm contrasts sharply with deep learning-based YOLO v4, providing a new perspective for measuring and evaluating the performance of single-board computers.
- **Suitability for Resource-Limited Environments**: The efficiency and classic features of the Canny algorithm make it highly suitable for running on resource-limited single-board computers, requiring minimal computational resources to perform efficient edge detection.

Through this comparative experimental setup, we can evaluate not only the performance of single-board computers in executing highly complex deep learning algorithms such as YOLO v4 but also measure their efficiency in performing mathematical computation-based image processing tasks like the Canny edge detection algorithm. This methodology allows us to comprehensively understand the applicability and limitations of single-board computers in different application scenarios, providing scientific evidence for selecting appropriate hardware platforms.

Development Environment and Tools

Programming Language (C Language)

- **Performance Optimization**: C language is renowned for its proximity to hardware operations and high-performance execution, making it an ideal choice for executing computationally intensive tasks such as image processing and neural network algorithms.
- **Widespread Usage**: C language is widely used in single-board computer and embedded systems development, providing a widely supported and mature development environment for this study.

Valgrind Tool

- **Memory Monitoring**: Valgrind is a programming tool used for memory debugging, memory leak detection, and performance analysis. In this study, Valgrind is used to monitor and evaluate the memory usage of single-board computers when executing image processing algorithms, which is crucial for performance evaluation.
- **Performance Analysis**: Through detailed reports provided by Valgrind, we can accurately understand the memory usage during algorithm execution, helping us to better evaluate the performance of different single-board computers.

ImageMagick Tool

- **Image Format Conversion**: ImageMagick is a powerful, stable, and widely used image processing tool that supports conversion between multiple image formats.

In this study, since the Canny edge detection algorithm requires BMP format input, while the datasets used in this study are in JPG and PNG formats, we use ImageMagick to convert images from the original dataset to the required BMP format.

- **Automation of Processing**: ImageMagick's batch processing capability automates and simplifies the process of converting a large number of images to the required format for experiments, ensuring consistency and accuracy of experimental data.

Through these carefully selected algorithms and tools, we can ensure the effectiveness and accuracy of the experiments, while providing a comprehensive performance evaluation to identify the best-performing single-board computers for executing image recognition tasks.

3.3 Criteria for Selecting Test Images

In this study, selecting suitable test images is crucial for evaluating the performance of single-board computers. We chose two datasets, VOCtrainval2012 and DIV2KtrainHR, as inputs for the experiment, based on several considerations:

3.3.1 Dataset Selection

- **VOCtrainval2012**: This dataset is known for its diversity and complexity of image content, containing a wide range of object categories and scenes, making it an ideal choice for evaluating the performance of image recognition algorithms. With an average image size of around 200KB, it represents smaller image samples, suitable for testing the efficiency and accuracy of single-board computers in handling lower-resolution images.

- **DIV2KtrainHR**: This dataset contains high-resolution images, with average large image sizes at 6MB and medium image sizes at 2MB, providing a way to test the ability of single-board computers to handle high-resolution images. Through this dataset, we can evaluate the performance of various single-board computers on more complex and data-intensive tasks.

The main reasons for selecting these two datasets are their high-quality image content, richness and complexity, as well as images of different sizes, which are critical factors in testing the processing capabilities of single-board computers.

3.3.2 Image Format Conversion

Since the Canny edge detection algorithm only accepts BMP format images, we need to convert the images from the VOCtrainval2012 and DIV2KtrainHR datasets to BMP format. This step significantly impacts the experimental design because BMP format

does not compress images, preserving all the details of the original image, but it also means that file sizes will significantly increase, requiring higher storage and processing capabilities of single-board computers. In this conversion process, we used the ImageMagick tool, which is a powerful, flexible image processing tool capable of efficiently handling large-scale image format conversions, ensuring the consistency and accuracy of the conversion process.

3.3.3 Sample Selection Method

Large and medium-sized images are taken from the DIV2KtrainHR dataset. The method chosen is to arrange the images in the DIV2KtrainHR dataset from largest to smallest and then select the largest 20 images as samples of large images and the smallest 20 images as samples of medium images. This method aims to assess the performance limits of single-board computers through images of different sizes. By selecting large images, we can test the ability of single-board computers to handle high-resolution, data-intensive images; by selecting medium-sized images, we can evaluate the performance of single-board computers in handling common-resolution images.

For small images, we randomly selected 20 images from the VOCtrainval2012 dataset to test the efficiency of single-board computers in handling low-resolution images. The advantage of this method is that it comprehensively evaluates the performance of single-board computers in different image processing tasks.

Through these criteria for selecting test images, we can comprehensively and deeply evaluate and compare the performance of different single-board computers in executing complex image recognition tasks, providing empirical evidence for selecting the most suitable single-board computer for efficient image recognition tasks.

3.4 Selection and Importance of the Valgrind Tool

In this study, we adopted the Valgrind tool to monitor and analyze the memory usage and program execution time of single-board computers when performing image processing tasks. Valgrind is a highly functional programming tool used for memory debugging, memory leak detection, and performance analysis, especially suitable for performance evaluation of complex software systems.

3.4.1 Performance Monitoring

Valgrind monitors memory fluctuations by taking memory snapshots rather than recording data at each time point. This means that it can capture the state of memory usage at key moments of program execution, such as program startup, before and after algorithm processing, and program termination, providing an effective way to observe and analyze peak and average memory usage. In addition, Valgrind can also monitor the program's execution time, which is crucial for evaluating processing efficiency and performance optimization.

Memory Usage

By monitoring the memory usage of single-board computers during the execution of image processing algorithms, we can evaluate their memory efficiency and identify any memory leak issues, which is particularly important for resource-constrained devices.

Program Execution Time

The program's execution time directly reflects processing efficiency and is one of the key indicators for evaluating algorithm performance.

3.4.2 Reasons for Selection

Several key reasons for choosing Valgrind as a performance monitoring tool include:

- **Functionality**: Valgrind provides a comprehensive toolset supporting various performance analysis and memory detection tasks, making it an ideal tool for comprehensive performance analysis.
- **Reliability**: As a mature tool, Valgrind is known in the development community for its stability and reliability, providing accurate and consistent performance evaluation results.
- **Recognition**: Valgrind is highly recognized among developers and is the preferred performance analysis tool for many open-source projects and commercial applications. This widespread usage and recognition ensure its effectiveness and applicability.

By using Valgrind as a performance monitoring tool, we can gain in-depth insights into the performance of single-board computers when executing image recognition and processing tasks, including memory usage and processing speed. This information is valuable for evaluating the performance of different single-board computers and guiding users in selecting the most suitable device for their application needs.

Chapter 4

Professional, Legal, Ethical, and Social Issues

4.1 Data Security and Privacy Protection

When discussing the technologies and algorithms used in this study, it is important to emphasize their openness and accessibility. This not only ensures the transparency of the research but also promotes the sharing and dissemination of scientific knowledge. Open source code and data are particularly crucial in our research as they enhance reproducibility and collaborative potential, thereby enriching the scientific community.

Here is further clarification on the open-source algorithms used in this study: This study primarily relies on two open-source algorithms: YOLO v4 and the Canny edge detection algorithm. The selection of these algorithms is based on their widespread application and outstanding performance in the field of image processing. Using these open-source tools not only accelerates the research process but also increases the flexibility and adaptability of experimental design, ensuring that our findings can be easily validated and extended by other researchers in the field.

Data security and privacy protection are major concerns when conducting image processing research. This study involves the use of multiple datasets for algorithm testing, including the publicly available datasets `VOCtrainval_2012` and `DIV2K_train_HR`, which mainly contain images without personally identifiable information. Nevertheless, we have taken several measures to ensure the security of the data and protect the privacy rights of participants:

Data Anonymization: Ensuring that all test images contain no information that can directly or indirectly identify individuals. **Access Contro**l: Restricting access to image data to project team members only. **Data Encryption**: Using strong encryption techniques to protect data in storage and transmission, preventing unauthorized access.

4.2 Use of Open Source Algorithms

When discussing the technologies and algorithms used in this study, it is important to emphasize their openness and accessibility. This not only ensures the transparency of the research but also promotes the sharing and dissemination of scientific knowledge. Here is further clarification on the open-source algorithms used in this study:

This study primarily relies on two open-source algorithms: YOLO v4 and the Canny edge detection algorithm. The selection of these algorithms is based on their widespread application and outstanding performance in the field of image processing. Using these open-source tools not only accelerates the research process but also increases the flexibility and adaptability of experimental design.

YOLO v4

- YOLO v4 is an advanced real-time object detection system implemented and maintained by AlexeyAB's Darknet project. The YOLO algorithm is widely recognized and used for its fast and accurate detection capabilities. - The source code of the YOLO v4 version used in this experiment is accessible on GitHub: https://github.com/AlexeyAB/darknet.git](https://github.com/AlexeyAB/darknet.git. This open-source project provides detailed implementation of the algorithm, usage instructions, and configuration guides, providing researchers with an easy-to-deploy and customizable object detection tool.

Canny Edge Detection Algorithm

- The Canny edge detection algorithm is a classic image processing algorithm widely used for extracting edge information from images. The version of the Canny algorithm used in this experiment comes from a publicly available open-source project, with the project address: https://blog.csdn.net/DK_csdn1/article/details/121798679?utm_source=miniapp_weixin - This project provides the implementation code and application examples of the Canny algorithm for researchers and developers, promoting the application and further optimization of the algorithm.

4.3 Ethical and Legal Considerations

When using open-source algorithms for research, it is essential to comply with the corresponding license agreements and ensure that research activities do not infringe intellectual property rights. In this study, when using these open-source tools, their license requirements have been strictly reviewed, and all research activities are ensured to be within the scope specified by these licenses.

Furthermore, in processing image data using these algorithms, this study strictly complies with data protection regulations, protecting the privacy and rights of data subjects. These practices not only reflect respect for professional ethics but also ensure the legal compliance of the research.

Through such methodology, this study aims to promote the reasonable use of open-source software while ensuring the ethical and legal integrity of scientific research activities.

4.4 Legal and Social Implications

The legal and social implications of this study involve the potential use of algorithms in practical applications, such as surveillance, autonomous vehicles, and personal devices. The image recognition technology in these application scenarios may involve monitoring of public and private spaces, raising concerns about privacy rights among the public. Therefore, this study strictly adheres to relevant data protection laws and standards in the design and implementation process, including the General Data Protection Regulation (GDPR) and other applicable national and regional laws.

4.5 Mitigation Strategies

To mitigate the above issues, this study adopts the following strategies:

- **Transparency**: Publicly disclose research objectives, methods, and expected use cases to increase project transparency and allow the public to understand the potential impacts of the technology. - **Ethical Review**: Conduct an ethical review before project design to ensure that the research complies with ethical standards and societal values.

4.6 Risk Assessment

The table below summarizes the main risks, likelihood, impact, and mitigation strategies that may be encountered during the course of this study:

Risk Description	**Likelihood**	**Impact**	**Mitigation Strategy**
Data or Source code loss	L	H	Keep data and source code backups in HWU cloud storage
Requirements change	M	H	Organize the project development in iterations
Project plan or scope change	M	H	Change the plan in coordination with the supervisor
Supervisor or Student is ill	L	H	Switch from face-to-face communication to Teams calls and emails

Table 4.1: Risk Analysis

By identifying and assessing these risks, this study has taken corresponding preventive measures to minimize potential negative impacts and ensure the smooth progress of the research.

Chapter 5

Experiment Design

5.1 Overview of Experimental Setup

5.1.1 Experimental Objectives

Currently, there is a lack of research in the field of single-board computers, especially regarding their performance in image processing. This experiment aims to comprehensively explore the performance of selected single-board computers in image processing. To ensure the value and reference significance of the experimental results, mainstream single-board computers available in the market are selected for evaluation. Additionally, widely acclaimed image processing algorithms—Canny edge detection algorithm and state-of-the-art YOLO v4 algorithm—are chosen. Both algorithms hold classical positions and have extensive applications in the field of image processing. This study seeks to determine the performance of different single-board computers in image processing tasks by comparing their execution time and memory usage.

5.1.2 Expected Goals

The main objective is to process specified images using pre-selected algorithms and obtain the execution time and memory usage for each task. Through comparative analysis, the aim is to clarify the image processing capabilities of single-board computers under controlled experimental conditions.

5.1.3 Basic Assumptions

Our basic assumptions are that the computing capability of the single-board computer's CPU (mainly reflected in the CPU clock frequency) and the size of the input image dataset will significantly affect the efficiency of image processing algorithms. Additionally, we assume that different categories of image processing algorithms, such as machine learning-based neural network image processing methods represented by YOLO v4 and traditional image processing methods represented by the Canny edge detection algorithm, will have differences in processing efficiency (evaluated based on the processing

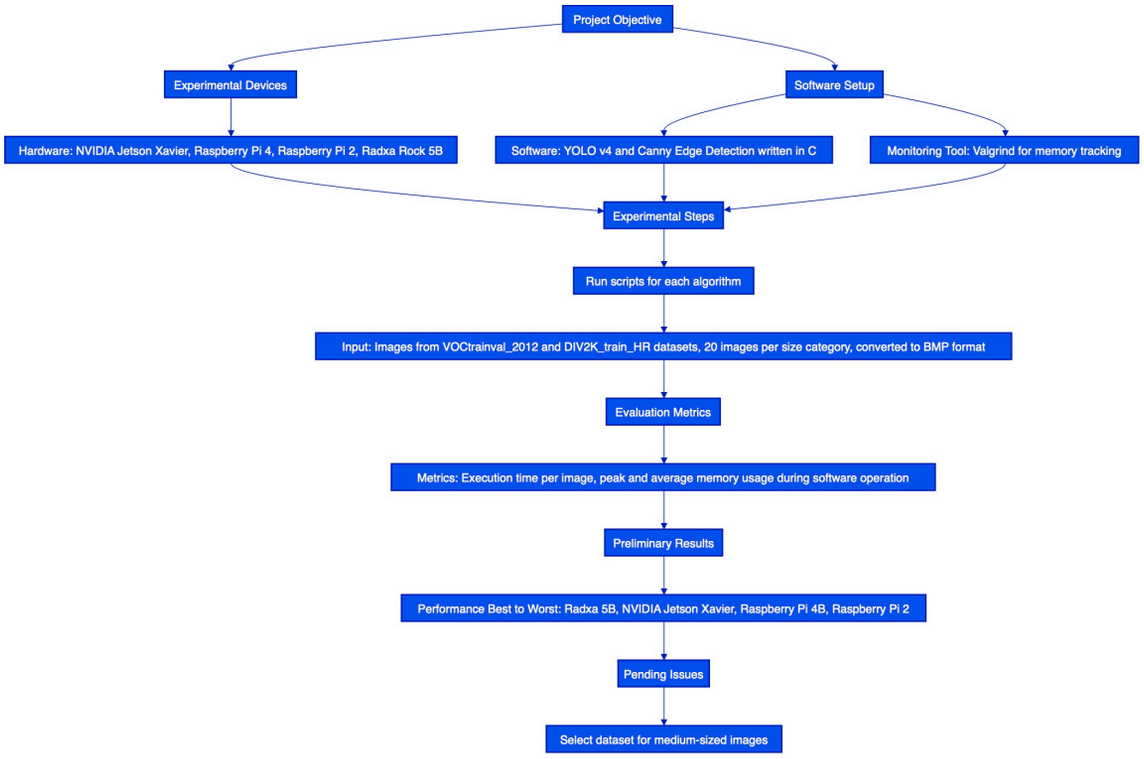


Figure 5.1: flow chart

time of a single image in the dataset in this study).

5.1.4 Theoretical Basis

The experimental design is based on existing research in computer vision and image processing, particularly the prominent positions of Canny edge detection and YOLO v4 in these fields. The choice of the Canny algorithm is because it represents computational efficiency in traditional image processing methods, while the selection of YOLO v4 as a deep learning method is based on its outstanding performance in processing speed and recognition accuracy. The selection of these two algorithms and their implementation on single-board computers aims to explore the performance of hardware when executing algorithms of different computational complexity. This comparison not only supports the current research but also advances the understanding of the role of hardware capabilities in the development of image processing techniques.

5.1.5 Flow chart for the project

5.2 Criteria for Selecting Single-board Computers and Their Justifications

To ensure the depth and breadth of the experiment, four representative single-board computers were carefully selected for comparative analysis, including NVIDIA Jetson Xavier NX, Raspberry Pi 4, Raspberry Pi 2B, and Radxa Rock 5B. The selection of these devices is based on their performance characteristics, market popularity, and the requirements of the controlled experimental design. This section will not reiterate the specific technical specifications of these devices, as they have been discussed in detail in Section 5, but will focus on describing the expected experimental performance of each device and their application scenarios in the research.

- **Raspberry Pi 2B**: As an older model in the control group, it is expected to contrast sharply with Raspberry Pi 4 in performance tests, demonstrating the improvement in single-board computer performance due to technological advancements.
- **Raspberry Pi 4**: As a device widely used in educational and hobbyist projects, it is expected to demonstrate its reliability and adaptability in increasingly complex tasks.
- **NVIDIA Jetson Xavier NX**: Known for its powerful graphics processing capabilities and AI algorithm optimization, it is expected to demonstrate excellent performance in image recognition tasks.
- **Radxa Rock 5B**: As one of the latest participants, its modern hardware is expected to demonstrate performance advantages compared to mature models.

5.3 Criteria for Selecting Test Images

This experiment selects two datasets, VOCtrainval_2012 and DIV2K_train_HR, as sources of test images. This choice is based on several key criteria:

1. **Data Diversity**: The VOCtrainval_2012 dataset, with its diverse object categories and background contexts, provides a wide range of test scenarios, ensuring a comprehensive evaluation of performance. This diversity is beneficial for assessing the ability of single-board computers to process complex images, ensuring the generality of the experimental results.
2. **Representativeness of Resolution**: The DIV2K_train_HR dataset contains high-resolution images, posing a significant challenge to the processing capabilities of single-board computers and effectively evaluating their performance under high data loads. Processing high-resolution images places high demands on CPU

and GPU computing capabilities and memory management, making it a focus of this experiment.

3. **Impact of Image Size**: Selecting images of different sizes aims to evaluate the performance differences of single-board computers in processing different image loads. The small-sized images provided by VOCtrainval_2012 contrast with the large-sized images in DIV2K_train_HR, helping to reveal the relationship between processing speed, memory usage, and image size.

4. **Applicability of Experimental Results**: Through these two widely used datasets, the experimental results will be more referenceable, assisting researchers and engineers in making more appropriate choices of single-board computers in practical applications.

5.4 Experimental Procedure

5.4.1 Preparation Phase

Setup of Test Environment Installation and Configuration of Operating Systems:

Before the experiment begins, it is necessary to ensure that all selected single-board computers have installed the operating system and been properly configured. Considering the consistency and reproducibility of the experiment, the following steps were taken to prepare the operating systems for different devices.

- **Raspberry Pi 2B and Raspberry Pi 4B**:

  I used a MacBook Pro with macOS Sonoma 14.4 operating system. Using the Raspberry Pi Imager provided by the official Raspberry Pi, I downloaded the installation program from the following link: https://downloads.raspberrypi.org/imager/imager_latest.dmg. The installer makes it easy to burn the operating system image to an SD card. Detailed instructions can be found on the Raspberry Pi official website https://www.raspberrypi.com/software/. After burning, Raspberry Pi 2B and Raspberry Pi 4B installed different versions of Raspberry Pi OS: Bullseye version for Raspberry Pi 2B and Bookworm version for Raspberry Pi 4B.

- **Radxa Rock 5B**:

  The operating system image chosen for Radxa Rock 5B is Kubuntu 22.04 version, downloaded from https://www.okdo.com/software-hub/. The specific image file is located at https://github.com/radxa-build/rock-5b/releases/download/b39/rock-5b_ubuntu_jammy_kde_b39.img.xz.

- **Nvidia Jetson Xavier NX**:

  Nvidia Jetson Xavier NX uses Ubuntu 18.04 version, obtained from Nvidia's developer website https://developer.nvidia.com/embedded/jetpack. The image

download link is https://developer.nvidia.com/downloads/embedded/l4t/r36_release_v2.0/jp60dp-orin-nano-sd-card-image.zip.

Due to the absence of dedicated flashing tools for Radxa Rock 5B and Nvidia Jetson Xavier NX, as found with the Raspberry Pi, we employed BalenaEtcher for image flashing. BalenaEtcher not only facilitates the flashing of operating system images onto SD cards but also verifies the data integrity post-flashing. The official website for this tool is https://etcher.balena.io, and it can be downloaded from https://etcher.balena.io/#download-etcher.

5.4.2 Preparation Stage

The preparation stage involved ensuring that all single-board computers were installed with the appropriate operating systems and configured similarly. This foundational step is crucial for the subsequent experimental processes.

5.4.3 Software Setup

The experiments utilized the Darknet framework and YOLO v4 for object detection, alongside the Canny edge detection algorithm for image processing. The following resources were utilized:

- Darknet: https://github.com/AlexeyAB/darknet.git
- YOLO Weights: https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v4_pre/yolov4.weights

5.4.4 Execution Stage

Each single-board computer processed all images from the same dataset using two different image processing algorithms. Adequate intervals between individual tests were ensured to avoid biases in performance measurement due to external factors, such as device overheating.

5.4.5 Monitoring Stage

The Valgrind tool was employed to capture and record the peak and average memory usage, as well as the processing time, for each algorithm processing each image. All data collection was automated to ensure consistency and reliability.

5.4.6 Data Processing

Post-experiment, the collected data were summarized and analyzed to extract the processing time and memory usage for each image, providing insight into the detailed and reliable image processing performance of each single-board computer.

5.5 Software Setup and Configuration

In this study, two algorithms were used: YOLO v4 and the Canny edge detection algorithm. Below are the configurations and operational steps for these two algorithms.

5.5.1 YOLO v4 Configuration

YOLO v4 is a deep learning-based object detection framework that is widely applied in real-time object detection tasks. The version of YOLO v4 used in this experiment was obtained from AlexeyAB's Darknet repository, available at: https://github.com/AlexeyAB/darknet.git.

1. **Darknet Environment Setup**: In the terminal, use the `cd` command to enter the directory that contains the compiled Darknet executable file, for example:

   ```
   cd ~/darknet
   ```

   For devices like Raspberry Pi, extract the Darknet archive into a designated directory, then enter this directory. Modify the Makefile to adapt to the configuration of the single-board computer (usually disabling GPU and CUDNN options). Execute the `make` command to compile Darknet.

2. **YOLO v4 Weights File Preparation**: Download the pre-trained YOLO v4 weights file to the Darknet directory. The weights file is available at: https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v4_pre/yolov4.weights.

3. **Running YOLO v4**: Run YOLO v4 for image object recognition using the following command format:

   ```
   ./darknet detector test <path_to_cfg_file> <
      path_to_weights_file> <path_to_data_file> <path_to_image>
   ```

   For images, Darknet will display detection results and draw bounding boxes on the images. The processed image is typically saved in `predictions.jpg`.

5.5.2 Canny Edge Detection Algorithm Configuration

The Canny edge detection algorithm is a classic edge detection method widely used in the field of image processing. The Canny algorithm used in this experiment comes from an open-source project, available at: https://blog.csdn.net/DK_csdn1/article/details/121798679?utm_source=miniapp_weixin.

1. **Algorithm Compilation**: First, write the source code file (`.c` file) and implement the function logic according to the declarations in the header file (`.h` file). Use the GCC compiler to compile the `.c` file into an object file (`.o` file). For example:

   ```
   gcc -c library.c
   ```

   This will generate an object file named `library.o`.

2. **Program Linking**: Link the object files into the final executable file. If there are multiple source files and object files, this step links them into a single executable. For example:

   ```
   gcc library.o -o canny
   ```

   This will create an executable file named `canny`.

3. **Executing the Algorithm**: Run `./canny` in the terminal to execute the Canny edge detection algorithm and perform related image processing.

The above configuration and installation steps ensure the consistency and reproducibility of the software environment used in the experiment. This is crucial for maintaining the accuracy and comparability of experimental results.

5.5.3 Application of Valgrind and ImageMagick

In the experiment, the use of Valgrind and ImageMagick tools was crucial for ensuring the accuracy and efficiency of testing.

Using Valgrind

1. **Performance Monitoring Setup**: For precise memory usage monitoring, we configured Valgrind specifically. When running the Canny algorithm, by specifying the Massif tool's output file, we could track memory allocation and deallocation:

```
1    valgrind --tool=massif --massif-out-file=/path/to/output.
   txt ./canny input_image.bmp output_image.bmp
2
```

   Important command-line options, such as `--massif-out-file`, are used to specify the location of the output file, which allows us to analyze later through the ms_print tool.

2. **Output data interpretation** Valgrind's Massif tool outputs a detailed memory usage report. In the report, the "Detailed snapshots" section identifies key points in memory usage, including peaks. As shown in the above output, the peak memory usage occurred in the 13th snapshot.For detailed valgrind output, see Appendix C (section .3).

3. **Performance Indicator Extraction** In the report, the "Detailed snapshots" section identifies key points in memory usage, including peaks. As shown in the output above, the peak memory usage occurred in the 13th snapshot.

   By summing the usage of all memory snapshots and dividing by the total number of snapshots, we were able to derive the average memory usage during the execution of the algorithm. Such a calculation provides a quantitative measure of the algorithm's memory efficiency.

Using ImageMagick

Using ImageMagick's command line tools, we performed batch image conversion to meet the format requirements of the experiment. This operation ensured the consistency of the input images in the experiment and enabled rapid conversion of large volumes of data.

The specific batch conversion command used to convert JPG format pictures to the BMP format required by the experiment is as follows:

```
for i in *.jpg; do convert "$i" "${i%.jpg}.bmp"; done
```

The command for converting PNG format pictures to BMP format is equally straightforward:

```
for i in *.png; do convert "$i" "${i%.png}.bmp"; done
```

During the conversion process, we ensured that all converted images maintained their original color depth and resolution, avoiding any image quality changes that could affect the experimental results.

Through these steps, Valgrind and ImageMagick became indispensable tools in the experimental process. Their use not only optimized the data processing workflow but also increased the accuracy and credibility of the experimental results. The comprehensive application of these two tools laid a solid foundation for subsequent data analysis and performance evaluation.

Batch Processing and Object Detection with YOLO v4

This script processes a set of images, applying the YOLO v4 algorithm for object detection while monitoring memory usage with Valgrind. For detailed implementation code, see Appendix A (section .1).

Automated Processing with the Canny Edge Detection Algorithm

This script demonstrates the automated running of the Canny edge detection algorithm on a set of images, with memory usage monitoring through Valgrind.For detailed implementation code, see Appendix A (section .2).

These scripts effectively automate the processing of image datasets and the monitoring of algorithm performance, establishing a robust foundation for subsequent data analysis and performance evaluation.

Chapter 6

Result Analysis

6.1 Performance Metrics

In this section, we explore the key performance metrics used in evaluating the performance of single-board computers. These metrics include runtime, peak memory usage, and average memory usage, which are crucial for understanding the processing capabilities of single-board computers. Each metric reflects the efficiency of system resource utilization and processing speed when executing specific tasks, providing direct guidance for selecting appropriate hardware configurations and optimizing algorithms.

Firstly, **runtime** is one of the fundamental metrics for measuring algorithm efficiency. It directly affects the system's response speed and user experience. In real-time systems, such as autonomous driving and surveillance systems, runtime is particularly critical because these systems require the rapid and precise processing of input data to make decisions in real time.

In appendixD you can look up the output of YOLOv4..4In this output:

```
/Users/wansiteng/Downloads/VOCdevkit/VOC2012/JPEGImages/2007
    _000129.jpg: Predicted in 7317.937000 milli-seconds.
```

This demonstrates the time YOLO v4 required to process the image. The predicted time "Predicted in 7317.937000 milli-seconds." indeed indicates the actual time spent by the model on detecting objects in the specified image. This time includes detecting objects in the picture, processing the image, and running the detection algorithm.

For the Canny edge detection algorithm, the **appendixE** provided the code way of how to record the running time .5.

After running, the runtime is output as follows:

```
Info: 500 x 375 x 24
gaussian_filter: kernel size 7, sigma=1
Canny Edge Detection took 0.972017 seconds.
```

Secondly, memory usage is another important performance metric. It includes peak memory usage and average memory usage, which are crucial because they directly relate to the system's resource requirements and stability when performing specific tasks, especially in resource-constrained environments. By using the Valgrind memory profiling tool, we can obtain detailed data on memory usage during program execution, providing us with the opportunity to deeply understand and optimize program performance.

Peak memory usage refers to the highest point of memory usage during program execution, reflecting the maximum resource demand. It is crucial for ensuring stable operation of the program on devices with limited resources. In experiments, by capturing memory snapshots with the Valgrind tool, we not only can identify the peak memory usage but also understand at which stage of program execution the highest memory demand occurs. For example, snapshot number 48 indicates the peak memory usage at a specific moment, which is a key data point in performance analysis and optimization.

Below is an example output from Valgrind, showcasing the memory usage analysis:

```
Number of snapshots: 72
Detailed snapshots: [1, 2, 3, 7, 11, 15, 27, 48 (peak), 50, 60, 70]
...
```

The Valgrind output captures memory usage at different points in time during the program's execution, helping to understand how memory usage changes. The "peak" snapshot, number 48, is particularly important as it indicates the maximum memory usage.

6.2 Average Memory Usage Calculation

To evaluate the average memory usage, the following mathematical formula was used:

$$\text{Average Memory Usage} = \frac{\sum_{i=1}^{n} \text{mem\_heap\_B}_i}{n} \tag{6.1}$$

where $\sum_{i=1}^{n} \text{mem\_heap\_B}_i$ represents the sum of all mem_heap_B values, and n is the number of these values. This calculation provides a rough indicator of the program's heap memory usage over its execution period.

6.3 Run time analysis

6.3.1 Runtime Analysis of the Canny Edge Detection Algorithm

This section presents the average runtime measurements for processing different types of images (large, medium, and small) using the Canny edge detection algorithm and YOLOv4 on various types of single-board computers. Due to significant performance differences among devices, both bar charts employ **logarithmic scales** to provide a clear visual representation. Moreover, because YOLOv4's runtime for a single image is

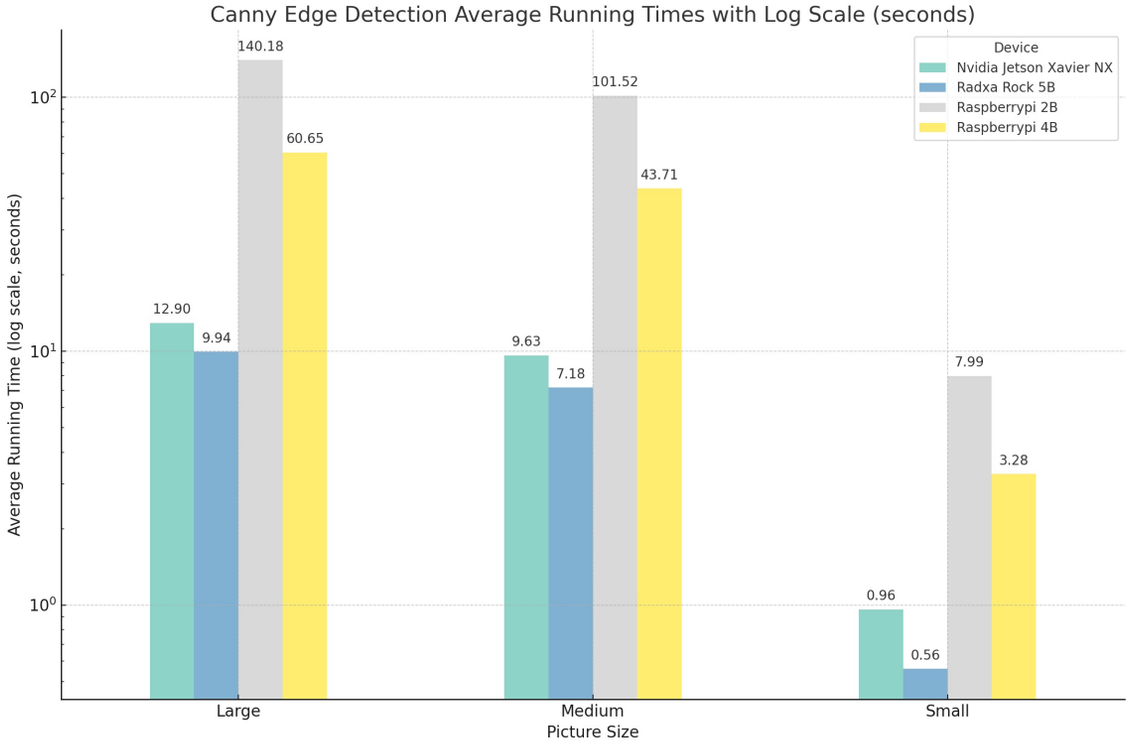


Figure 6.1: canny edge detection running times

relatively long, only two sizes of images are input in this case. Below are the experimental data:

Average Runtime of Canny Edge Detection Algorithm (seconds)

Device	**Small Images**	**Medium Images**	**Large Images**
Nvidia Jetson Xavier NX	0.96	9.63	12.90
Radxa Rock 5B	0.56	7.18	9.94
Raspberry Pi 2B	7.99	101.52	140.18
Raspberry Pi 4B	3.28	43.71	60.65

Device Performance Analysis

1. **Radxa Rock 5B** continues to exhibit the best performance across all categories, processing small images in just 0.56 seconds, medium images in 7.99 seconds, and large images in 12.90 seconds. This demonstrates Rock 5B's significant advantage in handling complex computations.

2. **Nvidia Jetson Xavier NX** performs similarly to Rock 5B for small images, but its processing times for medium and large images increase to 9.63 seconds and 60.65 seconds, respectively, showing a more pronounced performance decline with larger images.

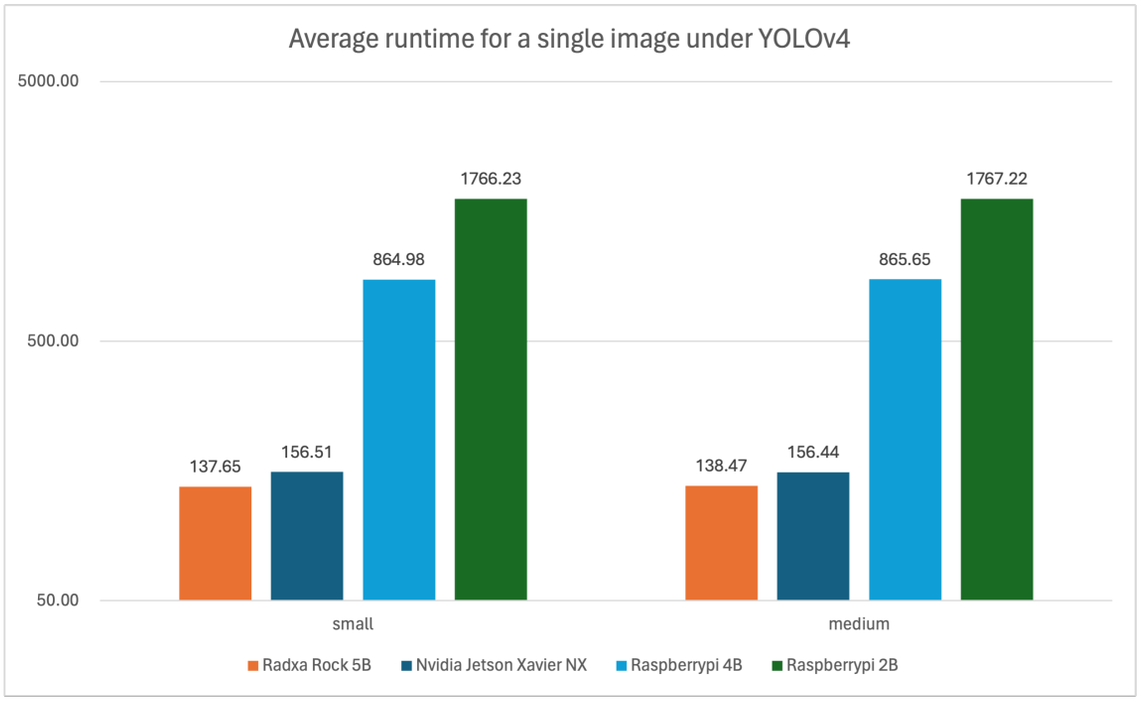


Figure 6.2: average runtime for a single image under YOLOv4

3. **Raspberry Pi 4B** shows a performance decline from 3.28 seconds for small images to 43.71 seconds for medium images and up to 101.52 seconds for large images. This reflects the most significant trend of performance degradation with increasing image size.

4. **Raspberry Pi 2B** performs the worst among all devices, taking 9.94 seconds for small images and up to 140.18 seconds for large images, indicating its limitations in handling data and compute-intensive tasks.

Runtime Analysis of YOLOv4

Average Runtime of YOLOv4 (seconds)

Device	Small Images	Medium Images
Nvidia Jetson Xavier NX	156.51	156.44
Radxa Rock 5B	137.65	138.47
Raspberry Pi 2B	1766.23	1767.22
Raspberry Pi 4B	864.98	865.65

This chart shows the average runtime of four different single-board computers (Radxa Rock 5B, Nvidia Jetson Xavier NX, Raspberry Pi 4B, and Raspberry Pi 2B) running the YOLOv4 algorithm on images of different sizes (small and medium).

Analysis Results:

1. **Radxa Rock 5B** has the shortest processing time for small-sized images at 137.65 seconds, demonstrating superior performance. For medium-sized images, its runtime increases to 864.98 seconds, but it still maintains the fastest level among all devices.

2. **Nvidia Jetson Xavier NX** takes slightly longer to process small images (156.51 seconds) and its time for medium images is also slightly longer than that of Rock 5B, at 865.65 seconds. This suggests that the Nvidia Jetson Xavier NX performs slightly below Rock 5B in running the YOLOv4 algorithm.

3. **Raspberry Pi 4B** performs well with small images (138.47 seconds) but its processing time significantly increases for medium images to 1766.23 seconds, which may indicate limitations in handling more complex image processing tasks.

4. **Raspberry Pi 2B** has the longest runtime in all categories, processing small images in 156.44 seconds and medium images in 1767.22 seconds. This aligns with its older hardware specifications and limited processing capabilities.

5. In a horizontal comparison of these four single-board computers, Radxa Rock 5B and Nvidia Jetson Xavier NX display similar runtime in processing YOLOv4 tasks, indicating very similar performance in deep learning tasks. Raspberry Pi 4B significantly lags behind the first two, but it is still about twice as fast as Raspberry Pi 2B. The runtime of Raspberry Pi 2B far exceeds the other devices, which is related to its hardware specifications and computing power.

Radxa Rock 5B and Nvidia Jetson Xavier NX significantly outperform the Raspberry Pi series in running the YOLOv4 algorithm, especially in processing medium-sized images. The performance of Raspberry Pi 4B and Raspberry Pi 2B decreases significantly with increased computational demands. These findings are important for guiding hardware choices for image recognition tasks that require running complex deep learning models. For applications that need to process images in a shorter time frame, Radxa Rock 5B might be the best hardware choice.

Furthermore, for **video processing applications**, the processing speed typically needs to be able to handle multiple frames per second. For instance, the standard video frame rate is 24 frames per second, meaning each frame's processing time should be less than 41.67 milliseconds (1000 milliseconds / 24 frames). However, in running the YOLOv4 algorithm, the processing time for a single small-sized image ranges from 137.65 seconds to 1766.23 seconds. **Clearly, even the best-performing Radxa Rock 5B far exceeds the 41.67 milliseconds per frame standard required for real-time video processing.** This indicates that while these single-board computers have certain capabilities in image processing, their processing speeds are not sufficient to support real-time video processing requirements of 24 frames per second.

For this reason, for tasks that require real-time processing of video to achieve real-time recognition of objects in videos, we need to consider improving hardware performance or finding alternative devices better suited for high-frequency processing to meet the application demands of real-time video processing.

6.4 Peak Memory Usage Analysis

The peak memory usage of the Canny edge detection algorithm during its execution is as follows:

	Nvidia Jetson Xavier NX	Radxa Rock 5B	Raspberry Pi 2B	Raspberry Pi 4B
Small picture	3,678,136.00	3,678,131.60	3,678,096.00	3,678,136.00
Medium picture	49,411,768.00	49,411,768.00	49,411,934.40	49,411,768.00
Big picture	64,765,009.60	64,765,009.60	64,765,201.60	64,765,009.60

The peak memory usage of YOLOv4 during its execution is as follows:

	Nvidia Jetson Xavier NX	Radxa Rock 5B	Raspberry Pi 2B	Raspberry Pi 4B
Small picture	1,609,366,400.00	1,609,369,360.00	1,609,366,400.00	1,609,366,400.00
Medium picture	1,650,460,680.00	1,650,460,680.00	1,650,460,680.00	1,650,460,680.00

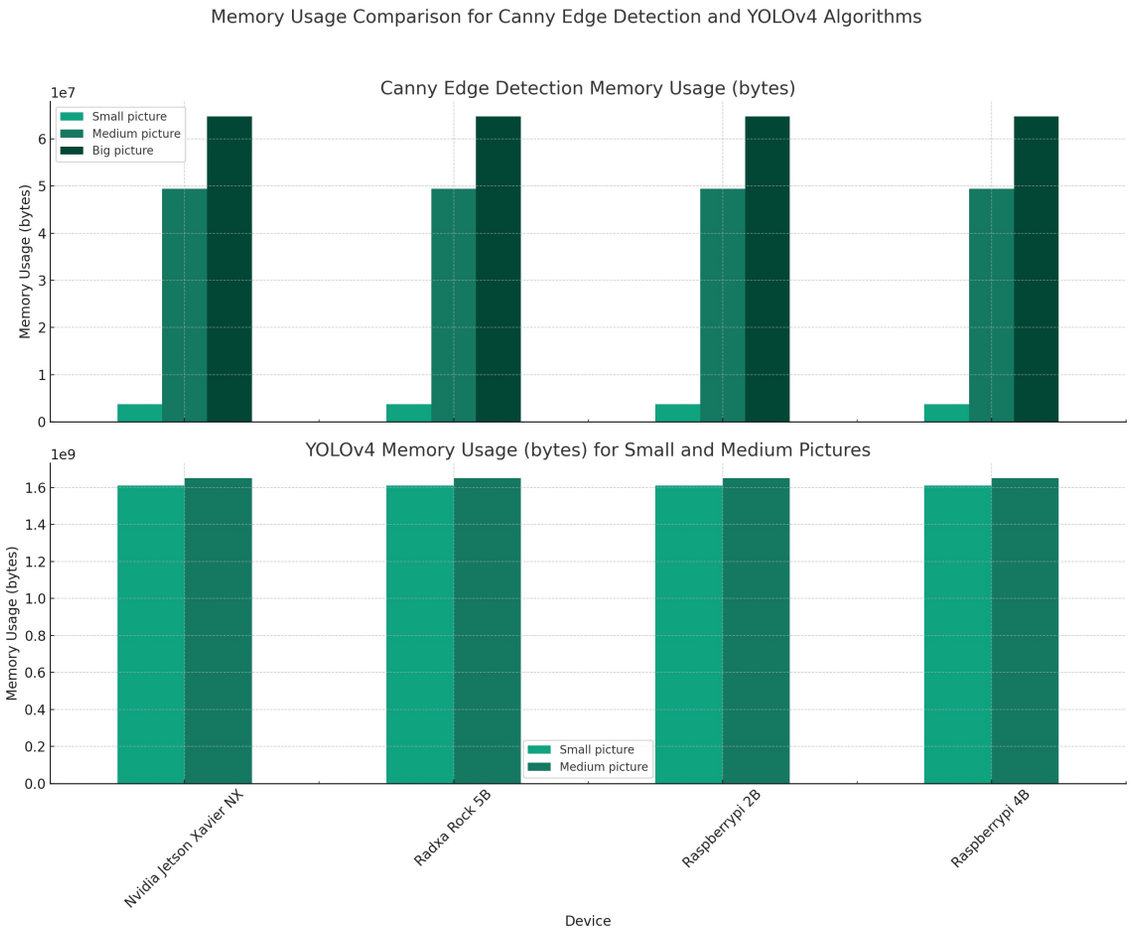


Figure 6.3: Peak memory usage

6.4.1 Peak Memory Usage Analysis

While analyzing the results, two phenomena were observed: why does the Canny edge detection algorithm show such a large variation in peak memory usage across different image sizes? There is a direct correlation between the input image size and peak memory usage – the larger the input image, the higher the peak memory usage; the smaller the input image, the lower the peak memory usage. Additionally, for YOLO, two vastly different image sizes showed no significant difference in peak memory usage. However, it is evident that the peak memory usage for small images is slightly lower than for medium images when input into YOLO.

The analysis of these phenomena can be understood through memory management and the implementation methods of the algorithms:

1. **Memory Usage Differences in the Canny Edge Detection Algorithm**
   The Canny edge detection algorithm involves several steps: noise reduction, gradient calculation, non-maximum suppression, double thresholding, and edge tracking. The steps that typically have the greatest impact on memory are gradient calculation and non-maximum suppression, which involve operations around each pixel, often including the storage of images twice (e.g., creating gradient maps, direction maps):

   - **Image Size and Memory Requirements**: The Canny algorithm calculates gradients for each pixel and may need additional memory to store intermediate results (like gradient magnitude and direction). As the image size increases, so does the number of pixels that need processing, and correspondingly, the memory needed to store this gradient and direction information increases. Thus, peak memory usage increases with the size of the image.

2. **Consistency of Memory Usage in the YOLO Algorithm**
   YOLO is a deep learning-based object detection algorithm that uses a fixed-size network architecture to process input images, which are first resized to a network-required fixed size (e.g., 416x416), regardless of the original dimensions of the input image. This means:

   - **Fixed-size Memory Requirements**: Since the YOLO network processes input images of a fixed size, memory usage is primarily influenced by the model structure (such as layers and filter sizes) rather than the original dimensions of the input images. This explains why different sizes of images, after resizing, have little impact on peak memory usage.
   - **Difference between Small and Medium Images**: Although all images are resized to the same dimensions, small images, compared to medium ones, may lose more details during resizing, which could lead to fewer computations (for example, fewer active neurons in feature maps). This might explain why memory usage for small images is slightly lower than for medium images.

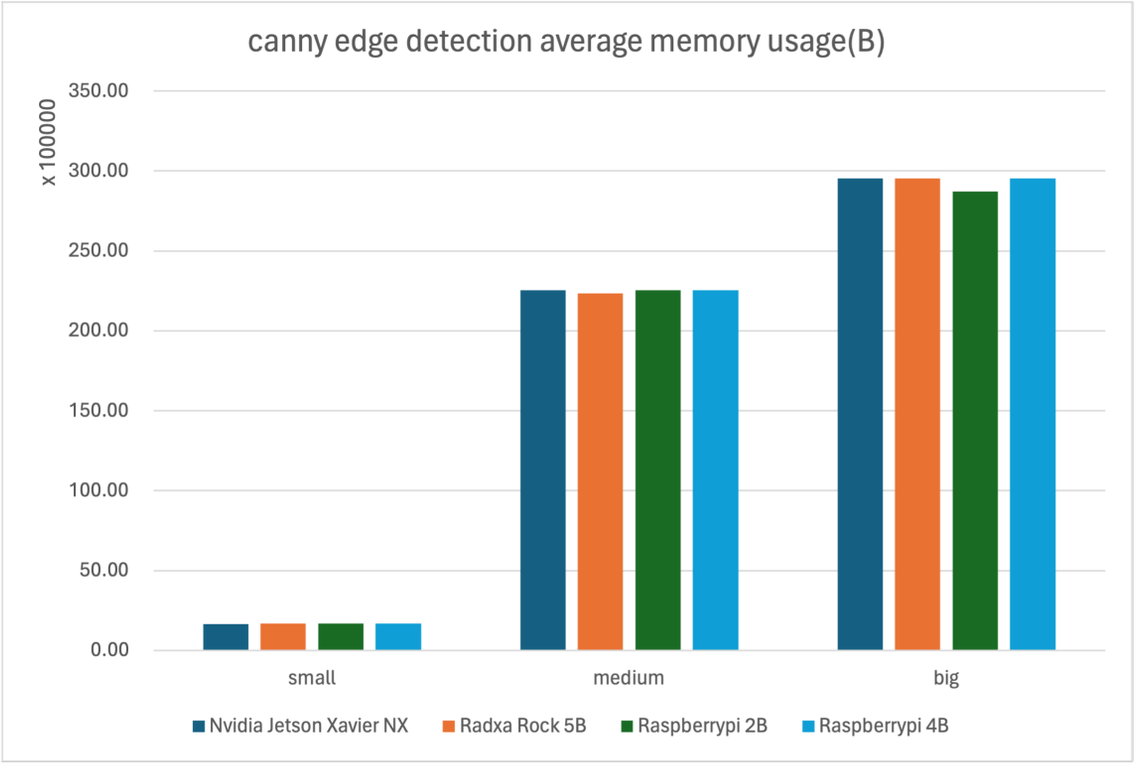


Figure 6.4: canny edge detection average memory usage

In summary, the memory usage of the Canny algorithm increases linearly with image size because the algorithm needs to compute and store extensive information for each pixel. In contrast, the memory usage of the YOLO algorithm is relatively fixed because it resizes any size image to fixed dimensions before processing, making memory demand primarily dependent on the network structure, not the original size of the input image.

6.4.2 Average Memory Usage Analysis

The average memory usage of the Canny edge detection algorithm during its execution is as follows:

	Nvidia Jetson Xavier NX	**Radxa Rock 5B**	**Raspberry Pi 2B**	**Raspberry Pi 4B**
Small picture	1,649,022.40	1,679,093.30	1,679,042.18	1,679,093.38
Medium picture	22,528,849.92	22,339,080.96	22,528,798.72	22,528,849.92
Big picture	29,530,607.66	29,527,975.68	28,712,614.64	29,527,975.68

Comparing the average memory usage and peak memory usage of the Canny edge detection algorithm reveals the following phenomena:

1. **Memory Usage Fluctuations**: Average memory usage is usually lower compared to peak memory usage, indicating that the algorithm does not always occupy the highest level of memory throughout the process, which may be related to the phased nature of data processing (e.g., loading, processing, and unloading phases).

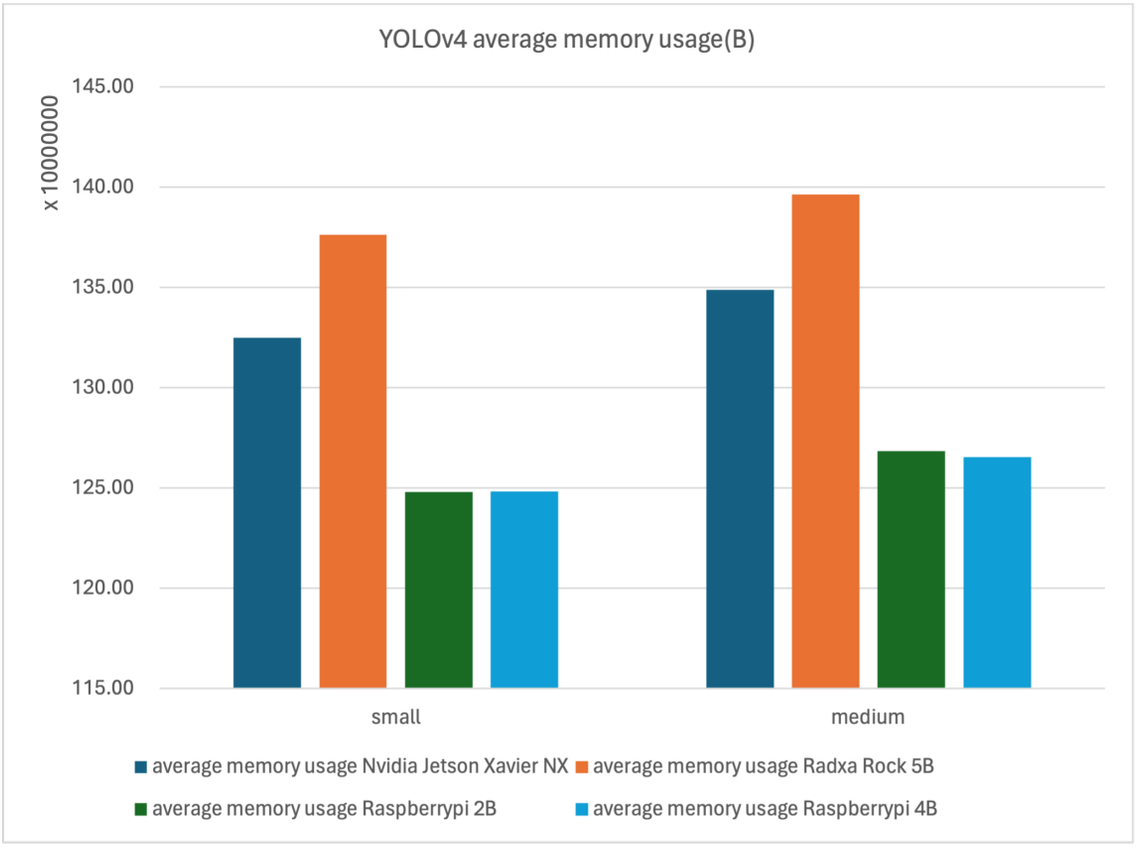


Figure 6.5: YOLOv4 average memory usage

2. **Consistency Across Devices**: Memory usage differences among devices are minimal across different sizes of images, indicating consistent memory management behavior of the algorithm on different hardware.

3. **Scale of Memory Usage**: All devices show significantly higher memory usage for large images compared to small and medium images. This is proportional to the amount of image data and indicates that memory demands are directly related to the volume of data being processed.

4. **Special Case**: The average memory usage of the Raspberry Pi 2B when processing large images is slightly lower than other devices, which may be related to its hardware configuration and memory management strategy.

The average memory usage of YOLOv4 during its execution is as follows:

	Nvidia Jetson Xavier NX	**Radxa Rock 5B**	**Raspberry Pi 2B**	**Raspberry Pi 4B**
Small picture	1,324,827,329.24	1,376,209,215.00	1,248,059,474.28	1,248,079,424.36
Medium picture	1,348,648,262.22	1,396,326,234.79	1,268,279,624.00	1,265,283,826.89

Comparing the average memory usage data with the peak memory usage data for the YOLOv4 algorithm provides the following insights:

1. **Comparison of Average and Peak**: The memory demand during busy moments, while average memory usage provides a more routine view of memory usage. This phenomenon indicates that, for most of the runtime, the YOLOv4 algorithm does not continuously occupy its peak memory.

2. **Differences Between Devices**: The Radxa Rock 5B generally has a higher average memory usage for small and medium images compared to other devices, suggesting differences in memory allocation or management. Raspberry Pi 4B and 2B have similar memory usage for small and medium images, indicating similar behavior under similar loads.

3. **Relation of Memory Usage to Image Size**: All devices use more memory for medium images than for small images, as expected, since larger images generally require more computational resources and memory.

6.5 Impact of Other Factors on Experimental Results

Temperature is an important consideration for the performance of single-board computers (SBCs), especially during high-intensity computational tasks such as image processing or running complex algorithms. Indeed, an increase in temperature can cause single-board computers to slow down.

6.5.1 Reasons for CPU Heat Production

1. **Resistive Heating**: When current flows through wires (typically copper or aluminum) inside a CPU, it encounters resistance. According to Ohm's Law, the current passing through a conductor generates heat, $P = I^2R$ (power = current squared times resistance). This heat, known as Joule heating, is a primary reason for the heating of CPUs and other electronic components.

2. **Transistor Switching**: Modern CPUs contain billions of transistors that open and close at very high frequencies to process data. Each time a transistor switches on or off, it consumes electrical energy and generates heat. This is due to the brief current leakage and energy consumed in redistributing charges during transistor switching.

3. **Voltage Drop**: In electronic devices, the voltage drop (the difference between the supply voltage and the actual voltage across components) also generates heat. At high operational frequencies, the internal voltage drop within a CPU may be more significant, leading to more heat production.

6.5.2 Impact of Temperature on Single Board Computer Performance

1. **Thermal Throttling**: When the temperature of a CPU rises to a certain level, to prevent damage, many modern processors automatically reduce their operating

frequency, a process known as thermal throttling or thermal slowdown. Throttling reduces the processor's power consumption and heat output but also lowers processing performance.

2. **Reliability Issues**: Operating at high temperatures for extended periods may reduce the reliability and lifespan of electronic components, as transistors and other micro-components may age prematurely due to thermal stress.

3. **Increased Error Rates**: High temperatures can increase the error rates of devices, as the conductive properties of electronic devices are significantly affected by temperature. Particularly in precision devices, even slight changes in temperature can lead to data corruption or operational errors.

6.6 Implications for Future Research

Broadening Device Testing

The current experimental design included several single-board computers, but future studies could diversify the selection of devices further. This could include emerging or specially designed single-board computers like Google Coral or Intel Movidius. Such inclusion would help in a more comprehensive evaluation of performance across different hardware.

Consideration of Environmental Factors

In applications such as drones and autonomous vehicles, devices often operate under harsh conditions like high temperatures and vibrations. Future experimental designs could simulate these environmental factors to test the stability and performance of single-board computers under such conditions.

Evaluation of Real-Time Processing Capabilities

Enhancing the evaluation of single-board computers' capabilities to process real-time data streams, such as video processing, is particularly crucial for drones and autonomous vehicles. Assessing their performance under continuous workloads can provide more practical data support for real applications.

Energy Efficiency Evaluation

Considering the resource-limited application environments, future research could delve deeper into the energy efficiency ratio (performance versus energy consumption) of single-board computers. This would not only aid in selecting superior-performance devices but also ensure optimal energy utilization.

Expansion of Application Scenarios

Drone Image Processing: Based on the conclusions of this experiment, the most suitable single-board computer can be selected for image recognition and processing tasks in drones. For instance, in surveillance or search and rescue missions requiring rapid processing and low-latency responses, devices with optimal memory management and processing speed should be chosen.

Autonomous Vehicles: The findings of this study could help automotive manufacturers decide which single-board computers are better suited for integration into driver-assistance systems for real-time image and sensor data processing. Considering the safety requirements of autonomous driving, choosing hardware that can operate stably in complex environments is particularly critical.

Environmental Monitoring and Disaster Response: In the fields of environmental monitoring and disaster response, single-board computers can be used to process data collected from various sensors, such as cameras and thermal imaging devices. Selecting appropriate devices based on the results of this experiment can optimize data processing workflows, enhancing response speed and accuracy.

Industrial Automation: In industrial automation, single-board computers can be applied in vision inspection systems for quality control, monitoring of automated assembly lines, and more. The experimental results can guide hardware selection in industrial applications, ensuring efficient and precise operation of production lines.

Through these specific improvement measures and perspectives on application scenarios, this study not only provides scientifically-based device selection recommendations but also promotes the widespread use of single-board computers in various practical applications, thereby advancing related technological progress and innovative applications.

.1 AppendixA

```
    # Define the directory variable where images are located
    for image in $(find ${IMAGE_DIR} -name "*.jpg"); do
        # If the number of processed images reaches 200, terminate
    the loop
        if [ $count -ge 200 ]; then
            break
        fi

        # Skip images whose filenames contain "._"
        if [[ $image == *._* ]]; then
            continue
        fi

        # Define the filename for Valgrind's output
        massif_out="massif_out_${count}.txt"

        # Run Valgrind's massif tool to monitor the memory usage
    of darknet YOLO v4, and filter out the lines containing "
    Predicted in"
        # This line includes the time required to process each
    image
        valgrind --tool=massif --massif-out-file=${massif_out} ./
    darknet detector test ${DATA_PATH} ${CONFIG_PATH} ${
    WEIGHTS_PATH} ${image} 2>&1 | grep "Predicted in"

        # Use the ms_print tool to convert massif's output into a
    readable format and save it
        ms_print ${massif_out} > "readable_${massif_out}"

        # Increment the image processing count by 1
        ((count++))
    done

    # Output the total number of processed images
    echo "processed $count images."

```

Listing 1: Shell Sample Script

.2 AppendixB

```
    #!/bin/bash

    # Define directories for input, output, and logs
    input_folder="/path/to/input"
```

```
    output_folder="/path/to/output"
    log_folder="/path/to/log"

    count=0

    for input_file in $(find ${input_folder} -name "*.bmp"); do
        filename=$(basename -- "$input_file")
        output_file="$output_folder/$filename"
        massif_out="${log_folder}/massif_out_${count}.txt"

        valgrind --tool=massif --massif-out-file=${massif_out} ./
    canny "$input_file" "$output_file"

        readable_massif_out_filename=$(basename -- "$massif_out")
        ms_print ${massif_out} > "${log_folder}/readable_${
    readable_massif_out_filename}"

        echo "Processed $input_file to $output_file"
        ((count++))
    done

    echo "Processed $count images."

```

Listing 2: Shell Sample Script

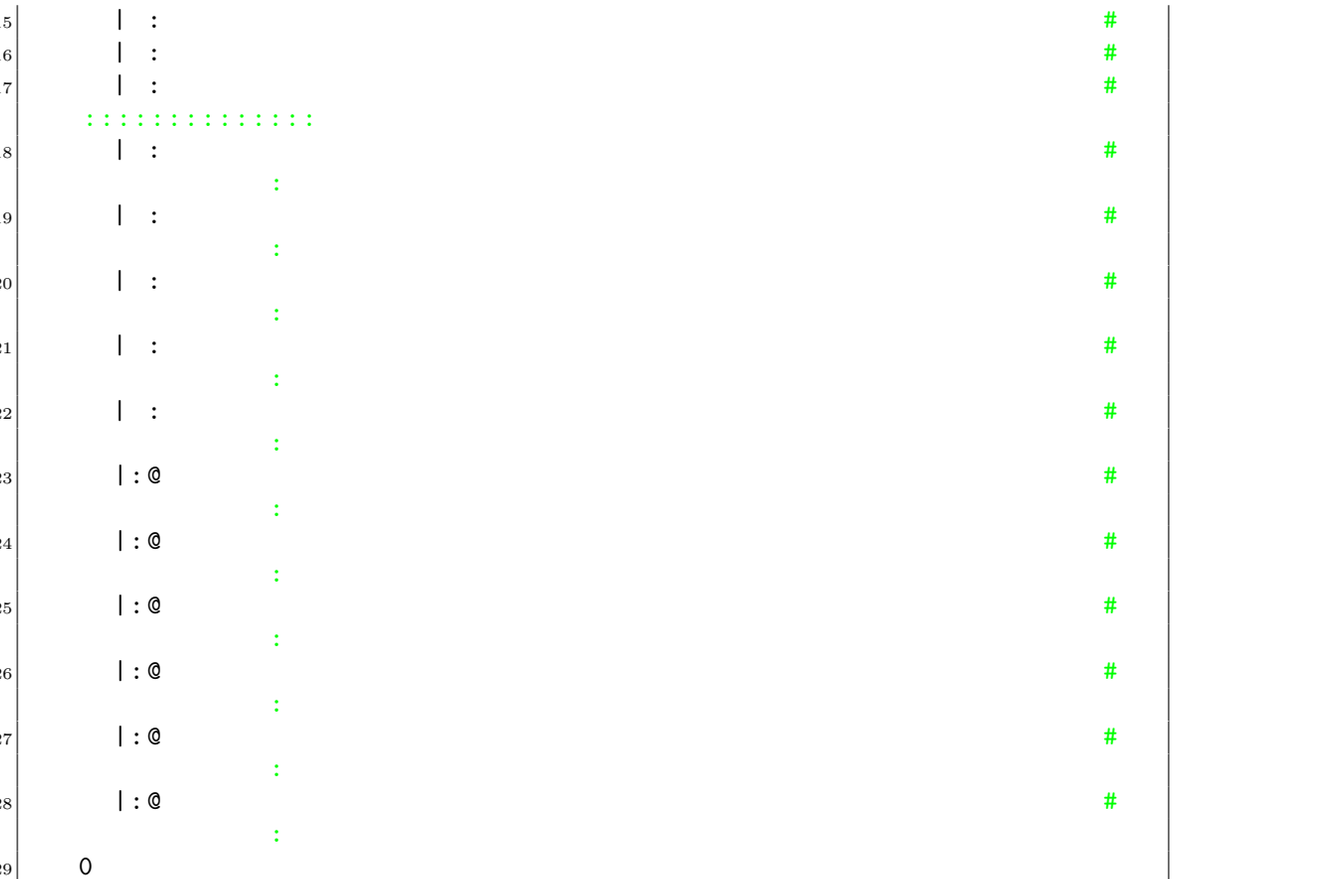
.3 AppendixC

```
    --------------------------------------------------------------------------------

Command:            ./canny /media/rock/SanDisk/BigPic/0002.png.
    bmp /media/rock/SanDisk/Canny/canny_output/0002.png.bmp
Massif arguments:   --massif-out-file=/media/rock/SanDisk/Canny/
    log/massif_out_0.txt
ms_print arguments: /media/rock/SanDisk/Canny/log/massif_out_0.txt
--------------------------------------------------------------------------------



    MB
71.91^ ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::#
     | :                                                                      #
     | :                                                                      #
     | :                                                                      #
     | :                                                                      #
     | :                                                                      #
```




```
Number of snapshots: 25
 Detailed snapshots: [4, 13 (peak), 23]

--------------------------------------------------------------------------------

  n        time(i)         total(B)   useful-heap(B) extra-heap(B)
      stacks(B)
--------------------------------------------------------------------------------

  0              0                0                0             0
                 0
  1        156,797              488              472            16
                 0
  2        157,450            8,688            8,664            24
                 0
  3        158,998       22,630,840       22,628,184         2,656
                 0
```

```
  4     656,714,562        22,630,840        22,628,184          2,656
                0
99.99% (22,628,184B) (heap allocation functions) malloc/new/new[],
    --alloc-fns, etc.
->99.95% (22,619,520B) 0x109073: load_bmp(char const*,
   bitmap_info_header_t*) (in /media/rock/SanDisk/Canny/canny)
| ->99.95% (22,619,520B) 0x10A583: main (in /media/rock/SanDisk/
   Canny/canny)
|
->00.04% (8,664B) in 1+ places, all below ms_print's threshold
   (01.00%)

```

Listing 3: valgrind output

.4 AppendixD

```
User
wansiteng@Edisons-MacBook-Pro darknet-master % ./darknet detector
   test cfg/coco.data cfg/yolov4.cfg yolov4.weights /Users/
   wansiteng/Downloads/VOCdevkit/VOC2012/JPEGImages/2007_000129.
   jpg
 GPU isn't used
 OpenCV isn't used - data augmentation will be slow
mini_batch = 1, batch = 8, time_steps = 1, train = 0
   layer   filters  size/strd(dil)      input
   output
   0 conv     32       3 x 3/ 1    608 x 608 x   3 ->  608 x 608 x
     32 0.639 BF
   1 conv     64       3 x 3/ 2    608 x 608 x  32 ->  304 x 304 x
     64 3.407 BF
   2 conv     64       1 x 1/ 1    304 x 304 x  64 ->  304 x 304 x
     64 0.757 BF
   3 route  1                                   ->  304 x 304 x  64
   4 conv     64       1 x 1/ 1    304 x 304 x  64 ->  304 x 304 x
     64 0.757 BF
   5 conv     32       1 x 1/ 1    304 x 304 x  64 ->  304 x 304 x
     32 0.379 BF
   6 conv     64       3 x 3/ 1    304 x 304 x  32 ->  304 x 304 x
     64 3.407 BF
   7 Shortcut Layer: 4,  wt = 0, wn = 0, outputs: 304 x 304 x  64
   0.006 BF

   //Partial output omitted

[yolo] params: iou loss: ciou (4), iou_norm: 0.07, obj_norm: 1.00,
    cls_norm: 1.00, delta_norm: 1.00, scale_x_y: 1.10
```

```
nms_kind: greedynms (1), beta = 0.600000
 151 route  147                                  ->   38 x  38 x 256
 152 conv    512       3 x 3/ 2     38 x  38 x 256 ->   19 x  19 x
    512 0.852 BF
 153 route  152 116                                ->   19 x  19
    x1024
 154 conv    512       1 x 1/ 1     19 x  19 x1024 ->   19 x  19 x
    512 0.379 BF
 155 conv   1024       3 x 3/ 1     19 x  19 x 512 ->   19 x  19
    x1024 3.407 BF
 156 conv    512       1 x 1/ 1     19 x  19 x1024 ->   19 x  19 x
    512 0.379 BF
 157 conv   1024       3 x 3/ 1     19 x  19 x 512 ->   19 x  19
    x1024 3.407 BF
 158 conv    512       1 x 1/ 1     19 x  19 x1024 ->   19 x  19 x
    512 0.379 BF
 159 conv   1024       3 x 3/ 1     19 x  19 x 512 ->   19 x  19
    x1024 3.407 BF
 160 conv    255       1 x 1/ 1     19 x  19 x1024 ->   19 x  19 x
    255 0.189 BF
 161 yolo
[yolo] params: iou loss: ciou (4), iou_norm: 0.07, obj_norm: 1.00,
    cls_norm: 1.00, delta_norm: 1.00, scale_x_y: 1.05
nms_kind: greedynms (1), beta = 0.600000
Total BFLOPS 128.459
avg_outputs = 1068395
Loading weights from yolov4.weights...
 seen 64, trained: 32032 K-images (500 Kilo-batches_64)
Done! Loaded 162 layers from weights-file
 Detection layer: 139 - type = 28
 Detection layer: 150 - type = 28
 Detection layer: 161 - type = 28
/Users/wansiteng/Downloads/VOCdevkit/VOC2012/JPEGImages/2007
   _000129.jpg: Predicted in 7317.937000 milli-seconds.
bicycle: 40%
person: 52%
person: 73%
bicycle: 29%
bicycle: 27%
person: 79%
Not compiled with OpenCV, saving to predictions.jpg instead

```

Listing 4: YOLOv4 output

.5 AppendixE

```
int main(const int argc, const char ** const argv)
{
    ...
    // Record the start time of the algorithm
    clock_t start_time = clock();
    ...
    // Record the end time of the algorithm
    clock_t end_time = clock();
    // Calculate and print the execution time of the algorithm
    double time_spent = (double)(end_time - start_time) /
    CLOCKS_PER_SEC;
    printf("Canny Edge Detection took %f seconds.\n", time_spent);
    ...
}
```

Bibliography

[Alaeddine et al., 2021] Alaeddine, H., Jihene, M., and Khemaja, M. (2021). An Efficient Deep Network in Network Architecture for Image Classification on FPGA Accelerator. In *2021 International Conference on Cyberworlds (CW)*, pages 72–77, Caen, France. IEEE.

[Andrews, 1976] Andrews, H. (1976). Image processing research.

[Ariza and Báez, 2021] Ariza, J. and Báez, H. (2021). Understanding the role of single-board computers in engineering and computer science education: A systematic literature review. *Computer Applications in Engineering Education*, 30:304 – 329.

[Ariza and Baez, 2021] Ariza, J. and Baez, H. (2021). Understanding the role of single-board computers in engineering and computer science education: A systematic literature review. *Computer Applications in Engineering Education*, page cae.22439.

[Bochkovskiy et al., 2020] Bochkovskiy, A., Wang, C.-Y., and Liao, H.-Y. M. (2020). YOLOv4: Optimal Speed and Accuracy of Object Detection. arXiv:2004.10934 [cs, eess].

[Choi et al., 2014] Choi, H., Son, D., Kim, C., and Kim, J. M. K. (2014). Impact of clock frequency and number of cores on gpu performance. *2014 International Conference on IT Convergence and Security (ICITCS)*, pages 1–4.

[Computer Vision Lab ETH Zurich,] Computer Vision Lab ETH Zurich. Div2k dataset. https://data.vision.ee.ethz.ch/cvl/DIV2K/. Accessed: 2024-04-06.

[Crusio et al., 2020] Crusio, W. E., Lambris, J. D., Lee, G. N., and Fujita, H. (2020). Deep learning in medical image analysis: Challenges and applications. *Deep Learning in Medical Image Analysis*.

[DK_csdn1,] DK_csdn1. Article on csdn. https://blog.csdn.net/DK_csdn1/article/details/121798679. Accessed: 2024-04-06.

[Esri,] Esri. How retinanet works. https://developers.arcgis.com/python/guide/how-retinanet-works/#:~:text=RetinaNet%20is%20one%20of%20the,with%20aerial%20and%20satellite%20imagery. Accessed: 2024-04-06.

[Han et al., 2013] Han, D., Zhang, T., and Zhang, J. (2013). Research and implementation of an improved canny edge detection algorithm. *Key Engineering Materials*, 572:566 – 569.

[Hao and Hao, 2020] Hao, S. and Hao, G. (2020). Research on oct image processing based on deep learning. *2020 IEEE 10th International Conference on Electronics Information and Emergency Communication (ICEIEC)*, pages 208–212.

[He and Xiong, 2021] He, D. and Xiong, S. (2021). Image processing design and algorithm research based on cloud computing. *J. Sensors*, 2021:1–10.

[Hegarty et al., 2016] Hegarty, J., Daly, R. G., DeVito, Z., Horowitz, M., Hanrahan, P., and Ragan-Kelley, J. (2016). Rigel: flexible multi-rate image processing hardware. *ACM Trans. Graph.*, 35:85:1–85:11.

[Holly et al., 2020] Holly, S., Wendt, A., and Lechner, M. (2020). Profiling Energy Consumption of Deep Neural Networks on NVIDIA Jetson Nano. In *2020 11th International Green and Sustainable Computing Workshops (IGSC)*, pages 1–6, Pullman, WA, USA. IEEE.

[Jayaraman et al., 2023] Jayaraman, T., Manimaran, B., Mani, S., and Joseph, M. J. (2023). Comprehensive Experimental Evaluation of Open Source Deep Learning Framework for Single Image Deraining Applications. In *2023 Fifth International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, pages 1–7, Erode, India. IEEE.

[Jovanović et al., 2014] Jovanović, P., Mileusnic, M., Pavić, B., and Mišković, B. (2014). Applications of the single board computers in the software defined radio systems. pages 882–886.

[Kim and Zabik, 1992] Kim, H. and Zabik, M. (1992). Design and evaluation of a versatile single board computer for embedded applications in scientific instrumentation. *Comput. Chem.*, 16:261–263.

[kuen Kim and Song, 2017] kuen Kim, Y. and Song, Y. (2017). Impact of processor cache memory on storage performance. *2017 International SoC Design Conference (ISOCC)*, pages 304–305.

[Lin et al., 2018] Lin, T.-Y., Goyal, P., Girshick, R., He, K., and Dollár, P. (2018). Focal Loss for Dense Object Detection. arXiv:1708.02002 [cs].

[Liu,] Liu, W. Ssd branch of caffe on github. https://github.com/weiliu89/caffe/tree/ssd. Accessed: 2024-04-06.

[Liu et al., 2016] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., and Berg, A. C. (2016). SSD: Single Shot MultiBox Detector. volume 9905, pages 21–37. arXiv:1512.02325 [cs].

[Manore et al., 2021] Manore, C., Manjunath, P., and Larkin, D. M. (2021). Performance of single board computers for vision processing. *2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 0883–0889.

[Moen et al., 2019] Moen, E., Bannon, D., Kudo, T., Graf, W., Covert, M., and Van Valen, D. (2019). Deep learning for cellular image analysis. *Nature Methods*, 16(12):1233–1246.

[Prasad and Sinha, 2011] Prasad, S. and Sinha, S. (2011). Real-time object detection and tracking in an unknown environment. In *2011 World Congress on Information and Communication Technologies*, pages 1056–1061, Mumbai, India. IEEE.

[Pundlik et al., 2017] Pundlik, S., Patil, V., Navalgund, M., Sahasrabudhe, A., and Shinde, S. (2017). Object tracking bot using on-board Jetson TK1: An approach to reduce communication overhead and time delay. In *2017 International Conference on Big Data, IoT and Data Science (BID)*, pages 65–72, Pune. IEEE.

[Rai, 2018] Rai, R. (2018). Applications of image processing. 1:15–17.

[Rani, 2017] Rani, N. (2017). Image processing techniques: A review. 5:40–49.

[Redmon, a] Redmon, J. Pascal voc dataset mirror. https://pjreddie.com/projects/pascal-voc-dataset-mirror/. Accessed: 2024-04-06.

[Redmon, b] Redmon, J. Yolo: Real-time object detection. https://pjreddie.com/darknet/yolo/. Accessed: 2024-04-06.

[Ren et al., 2016] Ren, S., He, K., Girshick, R., and Sun, J. (2016). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. arXiv:1506.01497 [cs].

[Shilpashree et al., 2015] Shilpashree, K., Lokesha, H., and Shivkumar, H. (2015). Implementation of image processing on raspberry pi. *International Journal of Advanced Research in Computer and Communication Engineering*, 4:199–202.

[S.K et al., 2022] S.K, P., Kesanapalli, S. A., and Simmhan, Y. (2022). Characterizing the Performance of Accelerated Jetson Edge Devices for Training Deep Learning Models. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 6(3):1–26.

[tao Jin, 2010] tao Jin, W. (2010). An improved canny edge detection algorithm. *Microcomputer Information*.

[Wikipedia, a] Wikipedia. Canny edge detector. https://en.wikipedia.org/wiki/Canny_edge_detector. Accessed: 2024-04-06.

[Wikipedia, b] Wikipedia. Central processing unit. https://en.wikipedia.org/wiki/Central_processing_unit. Accessed: 2024-04-06.

[Wikipedia, c] Wikipedia. Clock rate. https://en.wikipedia.org/wiki/Clock_rate. Accessed: 2024-04-06.

[Wikipedia, d] Wikipedia. Single-board computer. https://en.wikipedia.org/wiki/Single-board_computer. Accessed: 2024-04-06.

[Wnuk, 2008] Wnuk, M. (2008). Remarks on hardware implementation of image processing algorithms. 18:105 – 110.

[Yildirim et al., 2022] Yildirim, M., Karaduman, O., and Kurum, H. (2022). Real-Time Image and Video Processing Applications Using Raspberry Pi. In *2022 IEEE 1st Industrial Electronics Society Annual On-Line Conference (ONCON)*, pages 1–6, kharagpur, India. IEEE.

[Yu and Leeser, 2005] Yu, H. and Leeser, M. (2005). Optimizing data intensive window-based image processing on reconfigurable hardware boards. *IEEE Workshop on Signal Processing Systems Design and Implementation, 2005.*, pages 491–496.

[Zhang and Loidl, 2023] Zhang, J. and Loidl, H. (2023). F1TENTH: An Over-taking Algorithm Using Machine Learning. In *2023 28th International Conference on Automation and Computing (ICAC)*, pages 01–06, Birmingham, United Kingdom. IEEE.