T2R2 東京科学大学 リサーチリポジトリ
Science Tokyo Research Repository

論文 / 著書情報
Article / Book Information

題目(和文)	GPU を用いたコンピュータビジョンの高速化と最適化の研究
Title(English)	GPU-Based Acceleration and Optimization Research on Computer Vision
著者(和文)	WANGCHENYU
Author(English)	Chenyu Wang
出典(和文)	学位:博士(理学), 学位授与機関:東京工業大学, 報告番号:甲第12513号, 授与年月日:2023年9月22日, 学位の種別:課程博士, 審査員:遠藤 敏夫,坂本 龍一,髙邉 賢史,脇田 建,佐藤 育郎
Citation(English)	Degree:Doctor (Science), Conferring organization: Tokyo Institute of Technology, Report number:甲第12513号, Conferred date:2023/9/22, Degree Type:Course doctor, Examiner:,,,,
学位種別(和文)	博士論文
Type(English)	Doctoral Thesis

Powered by T2R2 (Science Tokyo Research Repository)

GPU-Based Acceleration and Optimization Research on Computer Vision

Department of Mathematical and Computing Science
School of Computing

Tokyo Institute of Technology

Chenyu Wang

Supervisor: Toshio Endo

Ph.D. Dissertation

June 2023

Abstract

In this dissertation, we present an in-depth study on GPU-based acceleration and optimization of computer vision task models, pursuing the trade-off between performance and speed of the model, and our research encompasses two primary research.

In our inaugural research, we focus primarily on the acceleration of the Single Shot MultiBox Detector (SSD), a model extensively adopted in object detection tasks. Given the computation-intensive nature of SSDs, we provide a comprehensive analysis of the challenges associated with performing their computations both effectively and efficiently. As a crucial part of our investigation, we proposed and implemented a solution for GPU acceleration. We successfully ported components previously computed on the CPU to the GPU and customized the model's components for optimal utilization of GPU resources. Our efforts particularly target the post-processing layer, which includes critical functions such as non-maximum suppression (NMS) and sorting. These functions, traditionally executed on the CPU, have now been skillfully migrated to the GPU. Our GPU-accelerated SSD model showcases superior detection speed without major sacrifices in object detection accuracy, achieving a superior trade-off between detection speed and model accuracy. This speed enhancement is accomplished by effectively leveraging the GPU's vast parallel processing capabilities and optimized memory management, translating into a substantial reduction in execution time. Our research underscores the significant potential and feasibility of using GPUs to accelerate object detection frameworks.

Next, we turn our focus to Transformer-based models, particularly the Swin Transformer. These architectures have proven to be remarkably effective in a wide range of computer vision tasks. However, their heavy computational requirements necessitate the implementation of efficient computing methods to enable their acceleration. Our objective is to harness the power of the GPU to expedite computations, all the while ensuring performance is preserved. In the original Swin Transformer architecture, we identify limitations, particularly concerning window information interaction when dealing

with large feature scales. To address these issues, we introduce the Pyramid Swin Transformer as an effective resolution. Although this refinement incurs some additional computational overhead, it adeptly manages the window information interaction problem intrinsic to large-scale feature dimensions. Consequently, our Pyramid Swin Transformer demonstrates commendable results across a range of computer vision tasks including image classification, object detection, semantic segmentation, and video recognition. This showcases the effectiveness, scalability, and versatility of our solution, justifying the computational trade-off involved. we also make research into GPU-based acceleration techniques for the Swin Transformer architecture. We propose innovative parallel processing strategies for window-based multi-head self-attention mechanisms, achieving substantial computational performance improvements. Our parallel computing methods proved to be effective, scalable, and versatile for various computational conditions, offering new insights for GPU-based deep learning acceleration.

Overall, this dissertation presents novel methods for accelerating and improving computer vision tasks on GPUs while demonstrating the adaptability and scalability of these techniques in the context of state-of-the-art computer vision architectures.

Acknowledgements

I would like to express my deepest gratitude to my esteemed advisor, Professor Endo, for welcoming me into his laboratory and providing invaluable guidance and support throughout my research journey. His mentorship has been a constant source of inspiration, and I am truly grateful for his unwavering belief in my potential. Professor Endo has not only offered his academic expertise but has also been a pillar of support in my personal life. Despite the significant differences between my previous research and my current endeavors, he has consistently offered tremendous help in selecting research topics, acquiring new knowledge, and navigating the challenges that have arisen during the course of my studies.

I would also like to extend my heartfelt appreciation to Professors Hirofuchi and Ikegami for their invaluable contributions to my research. Their insightful advice, constructive feedback, and expertise have greatly helped shape my work, enabling me to publish my research smoothly. The collaborations and discussions with them have enriched my understanding of the field and have fostered a conducive environment for intellectual growth and exploration.

Furthermore, I would like to acknowledge the contributions of my fellow lab members, who have provided a stimulating and supportive atmosphere throughout my research journey. Their camaraderie and encouragement have made my time in the laboratory an enjoyable and rewarding experience. I am grateful for the countless thought-provoking conversations and the shared moments of triumph and challenges that we have encountered together.

Lastly, I would like to express my profound gratitude to my parents for their unwavering love, support, and encouragement throughout my academic journey. Their sacrifices, understanding, and belief in my abilities have been instrumental in my pursuit of excellence in research. Their guidance and constant presence in my life have given me the strength and determination to overcome the obstacles I have faced and continue striving for success.

Contents

1 Introduction **9**
1.1 Motivation . . . 9
1.2 Problem Statement . . . 10
1.3 Contribution . . . 11
1.4 Thesis Structure . . . 12

2 Background **15**
2.1 Computer Vision . . . 15
2.1.1 Basic Computer Vision . . . 15
2.1.2 Machine Learning-based Computer Vision . . . 16
2.1.3 Deep Learning-based Computer Vision . . . 19
2.2 GPU . . . 31
2.2.1 GPU Architectures . . . 31
2.2.2 CUDA Platform . . . 35
2.2.3 High-Performance Computing with GPUs . . . 37

3 Speed-up Single Shot Detector on GPU with CUDA **40**
3.1 Introduction . . . 40
3.2 Related Work . . . 42
3.2.1 Object detection . . . 43
3.2.2 High Speed of Feature extraction . . . 45
3.2.3 Single Shot Multibox Detector (SSD) . . . 46
3.3 Our Implementation and Optimization . . . 50
3.3.1 Problem of Current Research . . . 50
3.3.2 Pre-Processing . . . 50
3.3.3 Feature Extraction Layer . . . 51
3.3.4 Proposal Layer . . . 52
3.3.5 Post Processing . . . 53
3.4 Result & Analysis . . . 56
3.4.1 Accuracy . . . 58
3.4.2 Speed . . . 61

3.5 Conclusions 68

4 Optimization for the Swin Transformer 70
4.1 Introduction 70
4.2 Related Work 73
4.2.1 Feature Pyramid Network 73
4.2.2 Vision Transformer 76
4.2.3 Swin Transformer 82
4.2.4 Mask R-CNN 84
4.2.5 UPerNet 86
4.2.6 Adam Optimization Algorithm 87
4.3 Pyramid Swin Transformer 88
4.3.1 Object Detection with FPN 93
4.3.2 Semantic Segmentation Head 93
4.3.3 Video Recognition Adaptations 94
4.3.4 Experiment and Result 95
4.4 High-speed Window-based Multi-head Self-attention 106
4.4.1 Our Method 107
4.4.2 Result & Analysis 112
4.5 Conclusion 116

5 Conclusion & Future Work 119
5.1 Conclusion 119
5.2 Future Work 120

A Publish List 138

List of Figures

2.1 Architecture of a Neural Network 19
2.2 A CNN architecture 20
2.3 Illustration of a pooling operation 25
2.4 VGG Net [115] 29
2.5 Residual Block[136] 30
2.6 NVIDIA A100 GPU architecture 32
2.7 GPU Architecture Unit[119] 33
2.8 Matrix Multiplication Operation with Tensor Cores[91] 38

3.1 Single Shot Multibox Detector Model (SSD512) 46
3.2 Non-maximum Suppression algorithm 48
3.3 Visualization of Map-reduce NMS proposal[92] 49
3.4 Post Processing Data Flow 55
3.5 Illustration of parallel merge sort with CUDA. 57
3.6 Pytorch version of SSD512 time ratio 62
3.7 Execution time Comparison Result 63
3.8 Execution time Comparison Result with Other Framework . . 68

4.1 Pyramid Swin Transformer 72
4.2 Transformer block 77
4.3 Self Attention Architecture 79
4.4 Multi-head Attention Architecture 80
4.5 Vision Transformer[15] 81
4.6 Shifted Window-based Self-Attention 83
4.7 Mask R-CNN[46] 85
4.8 UPerNet[129] 86
4.9 Swin Transformer architecture problem 89
4.10 Pyramid Swin Transformer architecture 90
4.11 Top-1 Accuracy vs FLOPs for Various Models 98
4.12 Object Detection on COCO 101
4.13 mAP vs FLOPs for Various Models(Mask R-CNN) 102

4.14 Semantic Segmentation on ADE20K 103
4.15 Parallel Window-based Multi-head Self-Attention 107
4.16 Parallelization Strategy 112
4.17 Comparison of Matrix Multiplication 113
4.18 Performance comparison with 8×8 window size 115

List of Tables

3.1 Test Environment 57
3.2 The mAP comparison of our SSD512 with the original 59
3.3 Execution time Of NMS and sorting(ms) 64
3.4 Execution time of Different NMS(ms) 65
3.5 Performances of NMS with different batch size(ms) 66

4.1 Pyramid Swin Transformer Detailed architecture 91
4.2 Test Environment 95
4.3 Results on Imagenet Image Classification 97
4.4 Results on COCO object detection with Mask R-CNN 100
4.5 Results on COCO object detection with Cascade Mask R-CNN 101
4.6 Results of ADE20K samantic segmentation with UperNet . . . 104
4.7 Results of Kinetics-400 video recognition. 105
4.8 Test Environment 112

Chapter 1

Introduction

1.1 Motivation

The rapid development of computer vision technologies in recent years has brought significant breakthroughs in various fields, including object detection, image classification, semantic segmentation, and video recognition [66, 105, 48]. These advancements have a wide range of applications, such as autonomous vehicles, robotics, surveillance, and medical imaging, among others. However, the growing complexity and computational requirements of state-of-the-art models pose challenges in terms of processing efficiency, energy consumption, and deployment in resource-constrained environments [44].

Graphics Processing Units (GPUs) have emerged as an effective solution to address the computational demands of deep learning models due to their high parallelism capabilities and superior performance in handling matrix operations [100, 20]. The utilization of GPUs for accelerating computer vision tasks has become increasingly important as researchers and practitioners strive to develop more sophisticated models that can deliver better performance while maintaining reasonable computational costs.

The initial motivation for this research arose from the observation that most existing acceleration-related studies primarily focus on deep neural networks, whereas other computationally-intensive operations in object detection frameworks remain largely unexplored. In particular, we noticed that many of these operations are executed on CPUs, leading to potential performance bottlenecks. This also prompted us to investigate the feasibility of leveraging GPUs to accelerate such operations, with the goal of validating the effectiveness of this approach, facilitating the trade-off between speed and accuracy.

Furthermore, our attention has also been paid to Transformer-based models due to their promising results across a range of computer vision tasks and the computational problems it urgently needs to solve, the Swin Transformer model being a notable example. Our preliminary investigation unveiled a latent parallelizable computation aspect within the Swin Transformer model. This discovery opens the possibility of employing GPUs to expedite the training and detection process of Swin Transformer models, which is promising potential to achieve a superior balance between accuracy and speed. However, in the course of our research, we identified several limitations within the Swin Transformer architecture [80]. This first led us to propose improvements, which produced promising results, further intensifying our curiosity in this domain. Then, we turned our attention towards studying GPU-based acceleration techniques for the Swin Transformer. Our primary objective was to leverage the potential parallelism inherent in the window-based multi-head self-attention mechanism, which is a core component of the Swin Transformer. By implementing GPU-based acceleration, we aim to enhance the processing speed of the Swin Transformer, thereby facilitating its deployment in practical applications. In doing so, we anticipate that our research will inspire new directions for further development and optimization of Transformer models in general, thus making contributions to the field of neural network architectures.

In summary, the motivation of this research lies in exploring and developing GPU-based acceleration and optimization techniques that can effectively enhance the performance of state-of-the-art computer vision models. By investigating efficient parallel processing approaches, optimization strategies, and algorithmic adaptations, we aim to bridge the gap between these advanced models and their real-world applicability, while simultaneously pushing the boundaries of current research in the field.

1.2 Problem Statement

Despite the remarkable progress in computer vision tasks, several critical issues hinder the real-world deployment and scalability of state-of-the-art models, particularly in the context of object detection and image classification. This research aims to address the following key problems:

1. **Computational Complexity:** Advanced computer vision models, such as deep neural networks and Transformer-based architectures, require substantial computational resources to process high-dimensional inputs and perform complex operations[48, 124]. This computational

complexity can lead to high latency and limit the applicability of these models in time-sensitive and resource-constrained scenarios.

2. **Limited GPU Acceleration Techniques:** While GPU-based acceleration has proven to be effective for deep learning tasks, the current literature primarily focuses on the acceleration of deep neural networks, with limited exploration of other computationally-intensive operations in object detection frameworks or other task frameworks. Moreover, the application of GPU-based acceleration techniques for Transformer models remains an underexplored area, further limiting the potential of these powerful architectures in real-world scenarios.

3. **Performance Bottlenecks:** Many object detection frameworks rely on CPU-based operations for several critical tasks, which can create performance bottlenecks and hamper the overall efficiency of the system[44]. Identifying and addressing these bottlenecks is crucial for enhancing the performance of object detection frameworks and facilitating their deployment in practice.

4. **Model Limitations and Improvement:** Existing Transformer-based architectures, such as the Swin Transformer [80], exhibit certain limitations that can hinder their performance on various computer vision tasks. Identifying these limitations and proposing improvements to the architecture can further enhance the capabilities of these models, potentially leading to better performance and more effective deployment.

To tackle these problems, this research focuses on the development and evaluation of GPU-based acceleration and optimization techniques for computer vision tasks. By exploring efficient parallel processing approaches, optimization strategies, and algorithmic adaptations, we aim to address the aforementioned challenges and contribute to the advancement of computer vision research and applications.

1.3 Contribution

In this dissertation, we present a comprehensive study on GPU-based acceleration of computer vision tasks, focusing on object detection and Transformer-based architectures. Our main contributions are as follows:

1. We propose a GPU-accelerated version of the Single Shot MultiBox Detector (SSD) for object detection, which improves the overall processing

speed and efficiency. By adapting the original algorithm for GPU execution, we demonstrate the feasibility and effectiveness of using GPU acceleration for object detection frameworks.

2. We identify limitations in the original Swin Transformer architecture and introduce the Pyramid Swin Transformer as an improved solution. These new architectures achieve superior performance across various vision tasks, including object detection, image classification, semantic segmentation, and video recognition.

3. We present a novel GPU-accelerated version of the window-based multi-head self-attention, leveraging the parallel capabilities of GPUs to improve the processing speed and efficiency of the computation. Our accelerated window-based multi-head self-attention demonstrates better performance than the original version, making it more suitable for real-world applications.

4. Through extensive experiments, we validate the effectiveness of our proposed techniques, showing that they consistently outperform state-of-the-art methods in their respective tasks. Our work provides valuable insights into the design of efficient and high-performing computer vision models, paving the way for future research and practical applications.

5. We conduct a thorough analysis of the computational bottlenecks in both object detection frameworks and Transformer-based architectures. This analysis helps to identify the key areas that can benefit the most from GPU acceleration, allowing us to focus our optimization efforts and maximize the performance gains. Our findings can serve as a valuable reference for researchers and engineers looking to optimize and accelerate their own computer vision models.

Our research not only contributes to the improvement of existing computer vision models but also offers new insights into the potential of GPU acceleration for a broader range of tasks. We anticipate that our findings will stimulate further exploration in this area and help drive the development of even more efficient and powerful computer vision solutions.

1.4 Thesis Structure

This thesis is organized into five major chapters, providing a detailed overview of the research undertaken on GPU-based acceleration and optimization of computer vision task models.

Chapter 1 sets the foundation for the research, presenting the motivation for exploring GPU-based acceleration and optimization of computer vision models, formulating the problem statement, and enumerating the contributions of this work. It outlines the fundamental challenges and aims that drive the thesis.

Chapter 2 delves into the background necessary to understand the concepts involved in the research. It starts with the basics of computer vision, progressing to machine learning and deep learning-based computer vision. It also provides an introduction to GPU architectures and the CUDA platform which are key components of the solutions proposed in the thesis.

Chapter 3, titled 'Speed-up Single Shot Detector on GPU with CUDA', delves into the optimization and acceleration of the SSD model, a popular choice for object detection tasks due to its balance of speed and accuracy. Given the computation-intensive nature of SSDs, we provide a comprehensive analysis of the challenges associated with performing their computations both effectively and efficiently. In this chapter, we present our innovative approach to accelerating the SSD model using GPU computation. We detail the process of porting the model's components, traditionally computed on the CPU, to the GPU, and customizing them for optimal utilization of GPU resources. Our efforts particularly target the post-processing layer, which includes critical functions such as non-maximum suppression (NMS) and sorting. By skillfully migrating these functions to the GPU, we are able to significantly reduce the execution time. Our GPU-accelerated SSD model showcases superior detection speed without major sacrifices in object detection accuracy, achieving a superior trade-off between detection speed and model accuracy. This speed enhancement is accomplished by effectively leveraging the GPU's vast parallel processing capabilities and optimized memory management. The results of our research underscore the significant potential and feasibility of using GPUs to accelerate object detection frameworks, paving the way for future advancements in this field.

Chapter 4, titled 'Optimization for the Swin Transformer', we shift our focus to Transformer-based models, specifically Swin Transformer. We provide a comprehensive review of related work including Feature Pyramid Network, Vision Transformer, and Swin Transformer, leading to finding the problem in Swin Transformer architecture and to our proposed methods and adaptations. And we improve the shortcomings of Swin Transformer architecture, presenting the Pyramid Swin Transformer which is based on Swin Transformer. The results and effectiveness of our model across various tasks like image classification, object detection, semantic segmentation, and video recognition are reported. Our novel framework performs better than Swin Transformer and achieves superior accuracy and computational complexity

trade-offs. We also cover our research on High-speed window-based multi-head self-attention which is the foundational operation of Swin Transformer. After examining the efficient attention mechanisms and matrix multiplication acceleration techniques from existing research, we detail our proposed methods for efficient matrix multiplication and parallelization strategies using GPU. We further implemented and evaluated this approach, and our findings indicate that leveraging GPU acceleration for window-based multi-head self-attention is a viable and effective strategy.

The final chapter, Chapter 5, encapsulates the conclusion of our research and suggests potential future work. It revisits the main findings from the previous chapters and proposes several directions for extending this research further. It marks the end of the main body of the thesis. This structured layout of the thesis will guide readers through the intricate journey of this research, from understanding the basic concepts to appreciating the depth and significance of GPU acceleration in computer vision task models.

Chapter 2

Background

2.1 Computer Vision

Computer vision is a field of study that focuses on enabling computers to understand and interpret visual information from the surrounding world, such as images and videos. The goal is to teach machines to process and analyze visual data in a way that is similar to how humans perceive their environment. Computer vision has a wide range of applications, including object recognition, image and video analysis, augmented reality, and autonomous driving.

2.1.1 Basic Computer Vision

Early computer vision techniques primarily relied on handcrafted features and simple image processing and analysis algorithms. These methods can be categorized into several key areas:

- **Edge detection**: This involves identifying the boundaries between different regions in an image, often based on changes in intensity or color. Popular edge detection techniques include the Sobel operator, which computes the gradient of the image intensity at each pixel using two 3x3 convolutional masks [116]; the Canny edge detector, which combines non-maximum suppression and hysteresis thresholding to find true edges [16]; and the Laplacian of Gaussian (LoG) operator, which combines Gaussian smoothing and the Laplacian operator to detect both strong and weak edges [86].

- **Image segmentation**: This is the process of partitioning an image into multiple regions, typically based on similarities in pixel values

or other properties. Common image segmentation techniques include thresholding, where pixels are classified based on their intensity values; region growing, which iteratively merges neighboring pixels with similar properties; and watershed segmentation, which treats image gradients as topographic relief and finds watershed lines separating adjacent regions [45].

- **Feature extraction**: These methods aim to extract meaningful information from images, such as corners, key points, and descriptors. Some well-known feature extraction techniques are Scale-Invariant Feature Transform (SIFT), which identifies and describes local features in images that are invariant to scale, rotation, and illumination changes [83]; Speeded Up Robust Features (SURF), an improved and faster version of SIFT that uses integral images and Haar-like wavelets for feature detection and description [6]; and Histogram of Oriented Gradients (HOG), which captures the distribution of gradient directions in an image to describe object shapes [28].

Although these approaches were effective for some specific problems, they generally lacked robustness and scalability, as they relied on manual feature engineering and could not adapt to new or more complex tasks.

2.1.2 Machine Learning-based Computer Vision

Machine learning-based computer vision methods focus on developing algorithms that can learn to recognize patterns and make decisions based on data without explicit programming. These methods rely on training models using large sets of labeled data, extracting features from images, and making predictions based on the learned patterns. Some of the key machine learning techniques that have been employed in computer vision tasks include:

- **Support vector machines (SVMs)**: SVMs are a widely-used supervised learning method for classification and regression tasks. In computer vision, SVMs have been extensively applied for image classification and object recognition tasks, often combined with hand-crafted features like Histogram of Oriented Gradients (HOG) [28] or Scale-Invariant Feature Transform (SIFT) [83]. SVMs aim to find the optimal decision boundary (i.e., the hyperplane in the high-dimensional feature space) that separates different classes, maximizing the margin between the classes. Despite their effectiveness in many computer vision tasks, SVMs can struggle with large-scale, high-dimensional data and may require complex feature engineering.

- **k-Nearest Neighbors (k-NN)**: The k-NN algorithm[65] is a simple and intuitive method for classification and regression tasks based on the concept of similarity between instances. In the context of computer vision, k-NN has been used for image classification, object recognition, and image retrieval. Given a query image, the k-NN algorithm finds the k most similar images in the training set (based on a chosen distance metric) and assigns the query image to the majority class of its neighbors. Although k-NN is easy to implement and understand, it may suffer from high computational complexity and storage requirements, particularly when dealing with large image datasets.

- **Boosting algorithms**: Boosting is a family of ensemble learning methods that combine the predictions of multiple weak classifiers to improve classification accuracy. AdaBoost (Adaptive Boosting) [39] is one of the most popular boosting algorithms and has been applied to various computer vision tasks, such as face detection [125] and object recognition. Boosting algorithms can be highly effective for object detection tasks, as they can achieve high detection rates with relatively low computational complexity. However, they may be sensitive to noisy data and may require careful tuning of hyperparameters.

- **Random forests**: Random forests[13] are ensemble learning methods that construct multiple decision trees and combine their predictions through averaging or majority voting. They have been used in various computer vision tasks, such as image classification[38], object detection, and semantic segmentation[114, 25]. Random forests can efficiently handle large datasets, provide robust and accurate predictions, and offer a degree of interpretability through the decision tree structure, provide robust and accurate predictions, and offer a degree of interpretability through the decision tree structure. However, they may struggle with high-dimensional data and may not be as effective as deep learning methods for certain tasks.

- **Graph-based methods**: Graphs provide a flexible and powerful representation for modeling complex relationships between image elements, such as pixels, regions, or objects. Graph-based methods have been widely used in computer vision for tasks like image segmentation, where the image is modeled as a graph with nodes representing pixels and edges representing spatial or feature-based similarities [37, 12]. Graph-cut algorithms can then be applied to partition the graph into disjoint regions corresponding to different image segments. For example, max-flow/min-cut can be used to find the cut with the minimum sum of edge

weights that divides the graph into disjoint sub-graphs. Other graph-based techniques, like Markov random fields and conditional random fields, have also been employed for various computer vision tasks, including object recognition, image restoration, and semantic segmentation [67, 106]. Graph-based methods can effectively model the relationships between image elements and incorporate domain-specific knowledge. However, they may suffer from high computational complexity, especially when dealing with large images or complex graphs.

- **Neural networks**: Before the advent of deep learning, shallow neural networks, like the one illustrated in Figure 2.1, were widely used for computer vision tasks[136]. These networks typically consisted of an input layer, one or two hidden layers, and an output layer. The input layer receives raw pixel data from the images, and each neuron in the hidden layer applies a non-linear transformation to the input data, allowing the network to learn complex patterns. This transformation can be represented as:

$$h_i = \sigma(\sum_j w_{ij}x_j + b_i) \tag{2.1}$$

where h_i is the output of the ith neuron in the hidden layer, x_j is the jth input, w_{ij} is the weight connecting the ith neuron to the jth input, b_i is the bias term for the ith neuron, and σ is the activation function [43].

The final output layer uses the transformed inputs from the last hidden layer to generate predictions, often through a softmax function for classification tasks, which can be represented as:

$$y_k = \frac{e^{z_k}}{\sum_j e^{z_j}} \tag{2.2}$$

where y_k is the output of the kth neuron in the output layer, and z_k is the input to the kth neuron [9].

Some popular shallow neural network architectures include the multi-layer perceptron (MLP) and the radial basis function (RBF) network [9]. The MLP is fully connected, meaning each neuron in a layer is connected to all neurons in the previous and next layers. The RBF network, on the other hand, typically has a single hidden layer and uses radial basis functions as activation functions, which can provide a measure of similarity to a set of exemplars.

Shallow neural networks have been applied to tasks such as handwritten digit recognition and face recognition [71]. Despite the success of these early neural network models in some computer vision tasks, they were limited by the lack of depth and the inability to learn hierarchical feature representations, which is a key strength of modern deep learning models. These limitations were addressed by the introduction of deep learning and convolutional neural networks, which we will discuss in the next section.

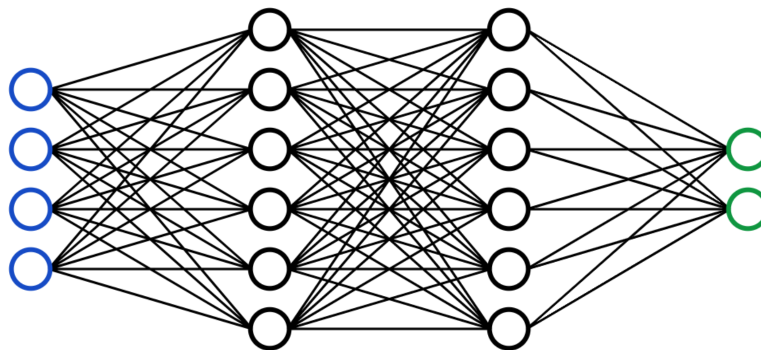

Figure 2.1: Architecture of a Neural Network

Machine learning-based computer vision techniques have played a significant role in the development of the field, paving the way for the rise of deep learning-based methods. These methods have addressed various challenges, including image classification, object detection, and segmentation, by learning from large labeled datasets and capturing meaningful patterns in the data [66]. However, many of these methods require hand-crafted features or carefully engineered representations, and they may struggle with large-scale, high-dimensional data. The advent of deep learning and convolutional neural networks has largely addressed these limitations, leading to breakthroughs in computer vision performance and enabling the development of more sophisticated models that can learn hierarchical feature representations directly from the raw data [43].

2.1.3 Deep Learning-based Computer Vision

Deep learning-based computer vision techniques have significantly advanced the field in recent years. Deep learning models, particularly convolutional

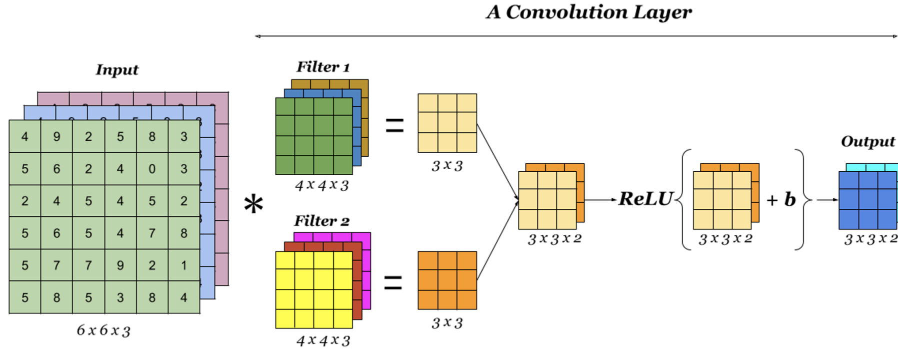

Figure 2.2: A CNN architecture

neural networks (CNNs), have demonstrated remarkable performance in various computer vision tasks, such as image classification [66, 115, 48], object detection [42, 105], semantic segmentation [82, 5], and video recognition [59, 123]. These advancements have led to widespread adoption and continued research in the field, driving the development of even more powerful and efficient computer vision models.

Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a class of deep learning models specifically designed for computer vision tasks[72]. The core concept of a CNN is the convolution operation, which allows the model to learn spatial hierarchies of features by applying a series of filters (also called kernels) to the input image.

The convolution operation is defined as:

$$(I * K)(i, j) = \sum_{m} \sum_{n} I(m, n) K(i - m, j - n), \tag{2.3}$$

In this formula, I is the input image and K is the kernel. (i, j) are the coordinates of a pixel in the image, and (m, n) are the coordinates of a pixel in the kernel. The operation computes the dot product of the kernel with each region of the input image, producing a new image where each pixel value represents the strength of the detected feature at the corresponding location in the input image.

An essential aspect of CNNs is their architecture, which typically consists of alternating convolutional layers and pooling layers, followed by one or more fully connected layers. The convolutional layers perform the feature

extraction, while pooling layers (such as max pooling or average pooling) reduce the spatial dimensions of the feature maps, and fully connected layers combine the learned features for the final classification or regression tasks. Activation functions, such as Rectified Linear Units (ReLU) [88], are applied after each convolutional or fully connected layer to introduce nonlinearity into the model.

CNNs are typically trained using backpropagation [107], which is an algorithm for minimizing the loss function by updating the model's parameters based on the gradient of the loss function with respect to each parameter. The gradient of the loss function is computed using the chain rule of calculus, which allows the gradient to be propagated backward through the layers of the network. The model's parameters are then updated using an optimization algorithm, such as stochastic gradient descent (SGD) [11], Adam [61]. The objective is to minimize the loss function, which measures the difference between the model's predictions and the ground-truth labels. Commonly used loss functions for classification tasks include the cross-entropy loss, while for regression tasks, the mean squared error loss is often used.

To prevent overfitting, various regularization techniques are employed, such as weight decay, dropout, and data augmentation. Weight decay adds a penalty term to the loss function, proportional to the squared norm of the model's weights. This encourages the model to learn simpler functions with smaller weights. Dropout is a technique where, during training, a fraction of the neurons in a layer are randomly "dropped" or deactivated, forcing the model to rely on a diverse set of features for its predictions. Data augmentation involves creating new training samples by applying random transformations, such as rotations, translations, and flips, to the original images. This increases the size of the training dataset and encourages the model to learn more robust features.

CNNs have been proven to be highly effective for various computer vision tasks, and their success can be attributed to their ability to learn hierarchical feature representations directly from raw data, their invariance to translations and other transformations [70], and their efficient use of parameters through weight sharing and pooling operations [115]. The development of advanced CNN architectures, such as VGGNet [115], Inception [120], ResNet [48], DenseNet [55], and MobileNet [53], has further pushed the boundaries of what is possible with deep learning-based computer vision approaches.

In summary, Convolutional Neural Networks have revolutionized the field of computer vision, providing state-of-the-art performance in a wide range of tasks. Their mathematical foundations, including convolution and pooling operations, as well as their efficient and robust learning capabilities, make them a powerful tool for processing and analyzing visual data.

Convolutional Layers

The input data is convolved in convolutional layers with a set of learnable filters (or kernels). The output of this operation, known as the feature map, is computed by sliding each filter across the input data and taking the dot product between the filter and the input data at each location.

Activation Functions Activation functions play a crucial role in deep learning architectures by introducing non-linearity into the model, enabling it to learn complex patterns and relationships among input features. These functions are applied element-wise to the neurons' outputs in a layer, transforming the weighted sum of inputs before passing it to the next layer. Several activation functions have been proposed and widely used in the literature, each with its own advantages and drawbacks. We will discuss some of the most popular activation functions below.

- **Sigmoid function:** The sigmoid function, also known as the logistic function, is one of the earliest activation functions used in neural networks. It is defined as:

$$f(x) = \frac{1}{1 + e^{-x}}, \tag{2.4}$$

  which maps the input value x to the range $(0, 1)$. The sigmoid function is smooth and differentiable, making it suitable for gradient-based optimization methods. However, it suffers from the vanishing gradient problem, where the gradients become too small during backpropagation, leading to slow convergence or training stagnation [50].

- **Tanh function:** The hyperbolic tangent (tanh) function is another widely-used activation function, defined as:

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \tag{2.5}$$

  which maps the input value x to the range $(-1, 1)$. Similar to the sigmoid function, the tanh function is smooth and differentiable, but it also suffers from the vanishing gradient problem [50].

- **Rectified Linear Unit (ReLU):** Introduced by Nair and Hinton [88], the ReLU function is currently one of the most popular activation functions in deep learning. It is defined as:

$$f(x) = \max(0, x), \tag{2.6}$$

which maps the input value x to the range $[0, \infty)$. ReLU is computationally efficient and mitigates the vanishing gradient problem, enabling the training of deeper networks. However, it can cause some neurons to become inactive during training, a phenomenon known as the "dying ReLU" problem, which may hinder the model's learning capacity.

- **Leaky ReLU:** To address the dying ReLU issue, Maas et al. [84] proposed the Leaky ReLU function, defined as:

$$f(x) = \max(\alpha x, x), \tag{2.7}$$

where α is a small positive constant (e.g., 0.01). Leaky ReLU allows small negative values, ensuring that neurons remain active during training and mitigating the dying ReLU problem. The dying ReLU problem refers to the phenomenon where some neurons become permanently inactive and only output 0. This can happen if a large gradient flows through a ReLU neuron, causing a large update to its weights. If the updated weights cause the weighted sum of inputs to that neuron to be negative, then the neuron will output 0. If this happens, then in subsequent iterations, the neuron will still output 0, no matter what values the input takes on.

- **Exponential Linear Unit (ELU):** Clevert et al. [21] introduced the ELU function to improve the ReLU's performance, defined as:

$$f(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha(e^x - 1), & \text{otherwise} \end{cases} \tag{2.8}$$

where α is a positive constant. ELU retains the advantages of ReLU, such as mitigating the vanishing gradient problem, and it allows negative outputs, ensuring that neurons remain active during training. ELU also has a smooth derivative for negative inputs, which can help with gradient-based optimization.

- **Swish:** Ramachandran et al. [101] proposed the Swish activation function, which is defined as:

$$f(x) = x \cdot \sigma(\beta x), \tag{2.9}$$

where σ is the sigmoid function, and β is a learnable parameter or a fixed constant. Swish is smooth, differentiable, and allows both positive and negative outputs. It has been shown to perform better than ReLU in some deep learning tasks [101].

Activation functions are essential components in neural networks, allowing them to learn complex, non-linear relationships among input features. The choice of activation function depends on the problem, architecture, and specific requirements of the task. Researchers are continually exploring new activation functions to improve the performance, stability, and convergence of deep learning models.

Pooling Layer The pooling operation is another key component of convolutional neural networks (CNNs). It is used to reduce the spatial dimensions of the feature maps while retaining the most important information. This operation helps to make the network invariant to small translations and reduces the computational complexity of subsequent layers [43].

The two most common types of pooling are max pooling and average pooling. Max pooling selects the maximum value within a specified window, while average pooling computes the average value within the window. The pooling operation can be defined as:

$$P(i, j) = \max_{m \in W_i, n \in W_j} I(m, n), \tag{2.10}$$

for max pooling, and

$$P(i, j) = \frac{1}{|W_i||W_j|} \sum_{m \in W_i, n \in W_j} I(m, n), \tag{2.11}$$

for average pooling, where I is the input feature map, P is the output pooled feature map, and W_i and W_j represent the pooling window dimensions.

It's worth noting that the pooling operation is applied independently to each input channel, and it does not change the number of channels [43]. This can be represented as:

$$P_c(i, j) = \max_{m \in W_i, n \in W_j} I_c(m, n), \tag{2.12}$$

for each channel c in the input feature map, where I_c is the cth channel of the input feature map, and P_c is the cth channel of the output pooled feature map.

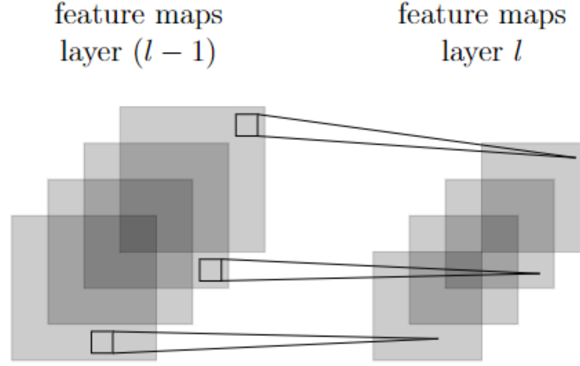

Figure 2.3: Illustration of a pooling operation

Fully Connected Layers Fully connected layers, also known as dense layers, are a fundamental component of various deep learning architectures. In these layers, each neuron is connected to every neuron in the previous layer, forming a complete bipartite graph between the two layers. The fully connected layer's primary function is to learn the final feature representations and perform high-level reasoning based on the features extracted by preceding layers, such as convolutional layers in the case of CNNs.

Given layer l as a fully connected layer, the output of the ith neuron in layer l can be computed as:

$$y_i^{(l)} = f\left(z_i^{(l)}\right), \tag{2.13}$$

where $f(\cdot)$ represents the activation function and $z_i^{(l)}$ is the weighted sum of the input neurons connected to the ith neuron in layer l. The weighted sum can be calculated as:

$$z_i^{(l)} = \sum_{j=1}^{m_1^{(l-1)}} w_{i,j}^{(l)} x_j^{(l-1)} + b_i^{(l)}, \tag{2.14}$$

where $m_1^{(l-1)}$ is the number of neurons in layer $(l-1)$, $w_{i,j}^{(l)}$ denotes the weight connecting the jth neuron in layer $(l-1)$ to the ith neuron in layer l, $x_j^{(l-1)}$ represents the output of the jth neuron in layer $(l-1)$, and $b_i^{(l)}$ is the bias term for the ith neuron in layer l.

Fully connected layers are often employed in the final stages of a deep learning model to combine the features extracted by previous layers and generate the final output, such as class probabilities in classification tasks or continuous values in regression tasks. It is worth noting that a fully connected layer intrinsically includes non-linearities through the activation function $f(\cdot)$, enabling the learning of complex patterns and relationships among the input features.

Normalization

Normalization techniques play a crucial role in deep learning models, including the Swin Transformer, by ensuring stable and efficient training. In this section, we will discuss the normalization methods used in the Swin Transformer and their underlying mathematical principles.

The two primary normalization techniques used in the Swin Transformer are Layer Normalization (LN) [4] and Batch Normalization (BN) [57]. While both methods aim to stabilize training by normalizing the input data, they differ in terms of the dimensions across which they compute the normalization statistics.

- **Layer Normalization:** Layer normalization (LN) is a normalization technique that is performed across each individual input or hidden layer of a neural network. Specifically, the mean and variance for normalization are computed across each single input data point, instead of computing across different input data points in the batch (as in batch normalization). For a given layer input $x = [x_1, x_2, ..., x_m]$, layer normalization computes the mean μ and variance σ^2 as follows:

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x_i \quad \text{(mean of the inputs)} \tag{2.15}$$

$$\sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu)^2 \quad \text{(variance of the inputs)} \tag{2.16}$$

After which, layer normalization applies the following transformation:

$$y = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} * \gamma + \beta \tag{2.17}$$

Here, x is the original input, μ and σ are the mean and standard deviation of the inputs, γ and β are learnable scale and shift parameters, ϵ is a small constant added for numerical stability, and y is the normalized output.

This type of normalization is particularly useful for models that need to maintain a consistent hidden state across each layer, as it doesn't normalize across the batch dimension.

- **Batch Normalization:** Batch normalization (BN) is a normalization technique that normalizes across batches of inputs, rather than across layers. For a given mini-batch $B = \{x_1, x_2, ..., x_m\}$ of size m, batch normalization computes the mean μ_B and variance σ_B^2 as follows:

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i \quad \text{(mean of the batch)} \tag{2.18}$$

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_B)^2 \quad \text{(variance of the batch)} \tag{2.19}$$

And then normalizes each input in the mini-batch with:

$$y = \frac{x - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} * \gamma + \beta \tag{2.20}$$

In this formula, x is the original input, μ_B and σ_B are the mean and standard deviation of the batch, γ and β are learnable scale and shift parameters, ϵ is a small constant added for numerical stability, and y is the normalized output.

The primary difference between BN and LN lies in the dimensions over which they compute the mean and variance. While batch normalization computes it over the batch dimension, layer normalization computes it over the feature dimension. This makes batch normalization dependent on the batch size and requires a large batch size to work effectively. In contrast, layer normalization works independently of batch size, making it more suitable for tasks where batch size might vary.

CNN Architectures

Over the years, numerous CNN architectures have been proposed, each improving upon its predecessors and pushing the limits of deep learning-based computer vision. In this section, we will review some of the most popular and influential CNN architectures.

- **AlexNet:** AlexNet [66], developed by Krizhevsky et al., was a major breakthrough in the field, achieving state-of-the-art performance in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012. AlexNet consists of five convolutional layers, followed by three fully connected layers and an output layer for classification. It introduced the use of ReLU activation functions and dropout layers to improve the training process and prevent overfitting.

- **VGGNet:** VGGNet, proposed by Simonyan and Zisserman [115], as shown in figure 2.4, is a significant milestone in the field of deep learning due to its simplicity, depth, and performance. VGGNet is characterized by its homogeneity and depth. It uniformly uses small (3x3) convolutional filters throughout the entire network, which was a shift from the previous architectures that typically used larger filters (e.g., 5x5 or 7x7) in the first convolutional layers. VGGNet demonstrated the importance of depth in neural networks for achieving high performance in computer vision tasks. Its architecture comprises several layers, with the main variants being VGG-16 and VGG-19, which consist of 16 and 19 layers, respectively. The depth of the network allowed it to learn more complex and abstract features, which contributed to its impressive performance. The network's layers follow a simple pattern: convolutional layers with small filters followed by max-pooling layers, with this sequence repeated multiple times to form the whole network. This simplicity and consistency make VGGNet easy to understand, implement, and modify, which has contributed to its wide adoption in the research community. Despite its higher computational cost and larger model size compared to other models like AlexNet or GoogLeNet, VGGNet has been widely adopted in various applications due to its excellent performance and simplicity. Its learned features have been shown to generalize well to other tasks and datasets, and its architecture has served as a foundation or a point of comparison for many subsequent models. Furthermore, the concept of using smaller filters in deeper networks, introduced by VGGNet, has been influential in the design of more recent architectures.

- **Inception (GoogLeNet):** The Inception architecture, also known as GoogLeNet [120], was developed by Szegedy et al. at Google. Inception introduced the concept of "inception modules," which consist of parallel convolutional layers with different filter sizes, followed by a concatenation of their outputs. This design enables the network to capture features at multiple scales. Inception also uses auxiliary classifiers

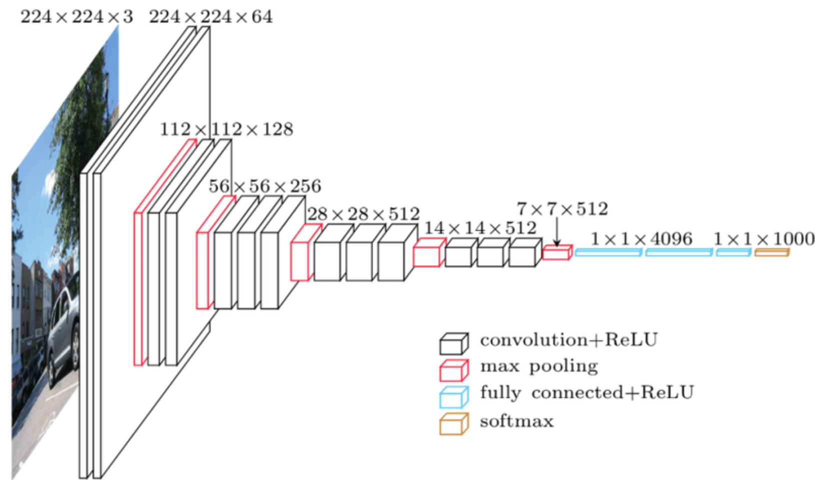

Figure 2.4: VGG Net [115]

during training to alleviate the vanishing gradient problem in deeper layers.

- **ResNet:** Residual Networks (ResNet), developed by Kaiming He et al., revolutionized the field of deep learning by introducing the concept of "residual connections" or "skip connections" [48]. In a traditional deep neural network, each layer learns a new representation of the data, and this new representation is passed on to the next layer. However, this approach becomes less effective as the network depth increases due to the problem of vanishing gradients. During backpropagation, the gradients often get smaller and smaller as they reach the initial layers, leading to slower convergence or the model not learning effectively. To solve this problem, ResNet introduced the idea of "shortcut connections" that allow the gradients to propagate directly through several layers by skipping intermediate layers. These shortcut connections perform identity mapping, and their outputs are added to the outputs of the stacked layers. This allows the stacked layers to fit a residual mapping instead of the original mapping, hence the name Residual Network. This innovative design made it feasible to train very deep networks (e.g., up to 152 layers), which was difficult with previous architectures. ResNet achieved state-of-the-art performance in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2015 competition, significantly reducing the error rate compared to previous models. It has since become a popular choice for many computer vision tasks, such as object detection, segmentation, and face recognition, due to its superior per-

formance and the generalization capability of the learned features. The principles behind ResNet have also influenced the design of subsequent deep learning architectures, making it a milestone in the development of deep learning models.

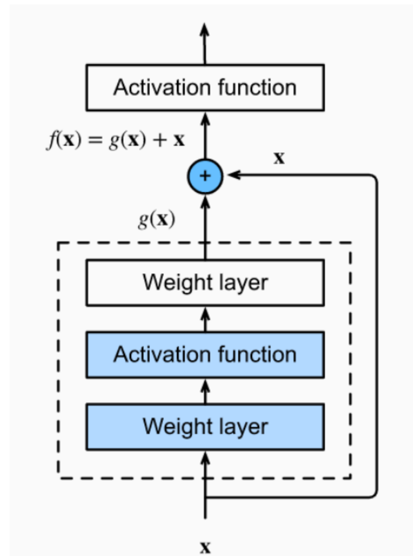

Figure 2.5: Residual Block[136]

- **DenseNet:** Dense Convolutional Networks (DenseNet) [55], proposed by Huang et al., extend the idea of skip connections by connecting each layer to every other layer in a feed-forward fashion. This design results in the efficient use of parameters and improved gradient flow. DenseNet has been shown to achieve high performance with fewer parameters compared to other architectures.

- **MobileNet:** MobileNet [53], developed by Howard et al., is designed for mobile and embedded vision applications, prioritizing efficiency and low computational cost. MobileNet introduces depthwise separable convolutions, which factorize standard convolutions into depthwise convolutions and pointwise convolutions, reducing the number of operations and parameters. MobileNet has multiple versions with varying trade-offs between performance and computational cost.

These popular CNN architectures have made significant contributions to the field of computer vision and have been widely adopted in various applications, ranging from image classification and object detection to semantic segmentation and image generation.

- **Capsule Networks:** Capsule Networks (CapsNets) [110], introduced by Sabour et al., is a novel architecture designed to address the limitations of traditional CNNs, particularly in regard to preserving spatial relationships between features. CapsNets replace scalar feature activations with vector-valued "capsules" that represent the presence and properties of entities in the input image. CapsNets have shown promise in tasks requiring the preservation of spatial hierarchies and pose estimation.

- **EfficientNet:** EfficientNet [121], proposed by Tan and Le, is an architecture that focuses on balancing model efficiency and accuracy by using a compound scaling method. The authors found that scaling up the width, depth, and resolution of the network simultaneously results in more efficient use of resources. EfficientNet has several variants (B0 to B7) with different capacities, achieving state-of-the-art performance while maintaining efficiency in terms of the number of parameters and computational cost.

The aforementioned CNN architectures have significantly influenced the field of computer vision, with wide applications in image classification, object detection, semantic segmentation, and image generation, among others. As advancements in CNN architectures continue to emerge and push the boundaries of what's possible in deep learning-based computer vision, researchers and practitioners alike are persistently exploring new techniques to improve these models' efficiency, robustness, and performance. These ongoing developments enable novel applications and the solving of increasingly complex problems within the realm of computer vision.

2.2 GPU

Graphics Processing Units (GPUs) have become an essential component in accelerating computer vision and deep learning applications due to their highly parallel architecture and superior computational capabilities compared to traditional Central Processing Units (CPUs). The following sections will provide a more detailed overview of GPU architectures, their components, and how they differ from CPUs.

2.2.1 GPU Architectures

A GPU is a specialized hardware designed to accelerate rendering tasks in graphics applications. However, their highly parallel architecture and pow-

erful computational capabilities make them well-suited for general-purpose parallel computing, including computer vision and deep learning tasks. GPUs are composed of thousands of smaller cores that can execute many threads simultaneously, making them ideal for data-parallel tasks. Unlike CPUs,

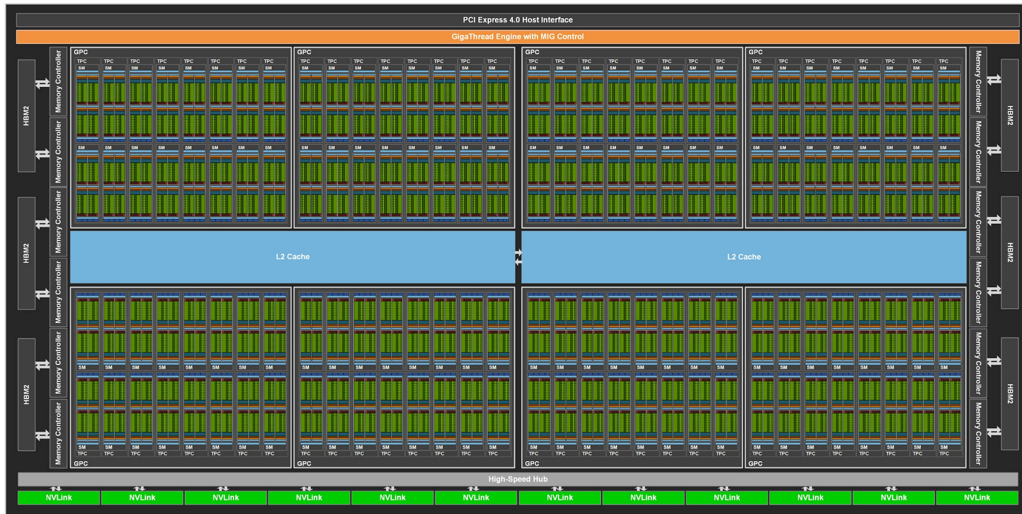

Figure 2.6: NVIDIA A100 GPU architecture

which typically have a small number of cores optimized for single-threaded performance and complex control flow, GPUs have a massive number of simpler cores designed for parallelism and throughput-oriented tasks. This architectural difference makes GPUs more suitable for processing large-scale, high-dimensional data common in computer vision and deep learning applications.

NVIDIA is one of the leading manufacturers of GPUs, and their GeForce, Quadro, and Tesla product lines are widely used in research and industry. NVIDIA GPUs are based on several architectures, such as Fermi, Kepler, Maxwell, Pascal, Volta, Turing, and Ampere [93]. Each successive architecture has introduced improvements in performance, energy efficiency, and programmability.

As illustrated in Figure 2.6, a typical GPU architecture consists of several streaming multiprocessors (SMs), each containing multiple CUDA cores (also called streaming processors, or SPs) for arithmetic operations, special function units (SFUs) for complex mathematical functions, and load/store units (LD/ST) for memory access.

GPUs have different types of memory, each with varying access speeds and capacities:

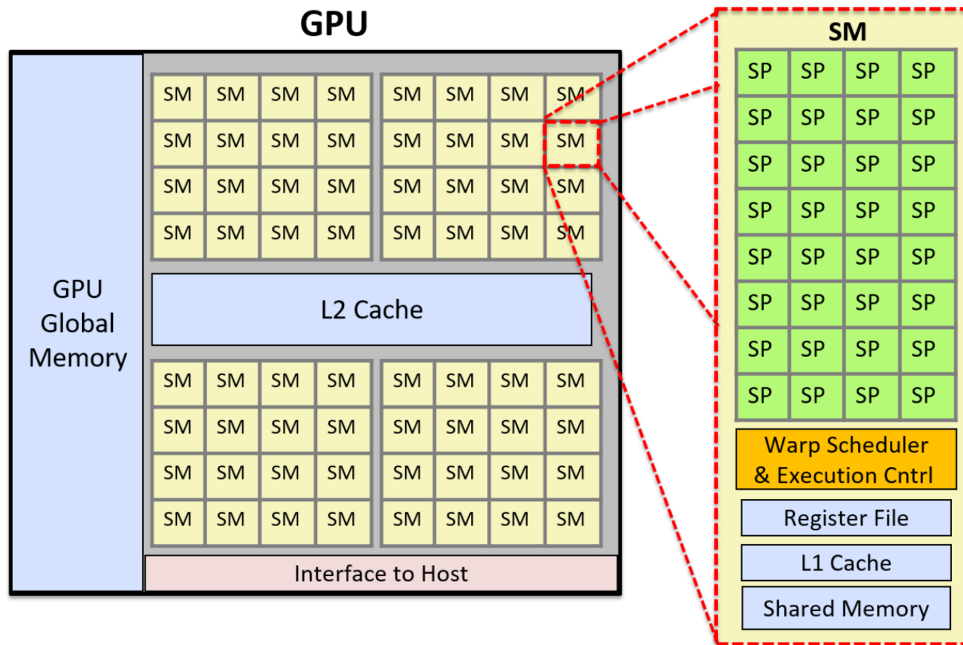

Figure 2.7: GPU Architecture Unit[119]

- **Global memory** is the largest memory type on a GPU and is accessible by all threads and the host (CPU). It is also known as device memory and is allocated off-chip, which makes it slower compared to other memory types. However, global memory provides high bandwidth and large capacity, making it suitable for storing large data sets.
- **Shared memory** is a smaller, faster memory type that is shared among the threads within a thread block. It is allocated on-chip, providing low-latency access for threads. Shared memory is used for inter-thread communication and to cache frequently accessed data, reducing the need for slower global memory accesses.
- **Constant memory** is a read-only memory type that can be cached for faster access. It is used for storing constant values, such as model parameters or lookup tables, which remain unchanged throughout the execution of a kernel. Constant memory is also allocated off-chip but can be cached on-chip for improved access speed.
- **Local memory** is a private memory space for each thread, used for storing local variables and temporary values that do not fit into the

thread's registers. Local memory is allocated off-chip and has similar access times as global memory.

- **Texture memory and surface memory** are specialized memory types for handling 2D and 3D data, such as images and volumes. These memory types offer specific hardware features, such as filtering and boundary handling, which can be useful in computer vision applications. Texture and surface memory are also cached for faster access.

Understanding the characteristics and usage of these different memory types is crucial for optimizing GPU performance in computer vision and deep learning applications. Efficient memory management and access patterns can significantly improve the execution speed and reduce the memory footprint of GPU-accelerated algorithms.

In addition to memory types, other architectural components also play a significant role in GPU performance. For instance, the number of registers per thread, the number of threads per block, and the occupancy of streaming multiprocessors (SMs) can influence the overall throughput and efficiency of a GPU-accelerated application. Balancing these factors to utilize the available hardware resources fully is a key aspect of optimizing GPU-based algorithms[119].

Another important difference between GPUs and CPUs is their approach to instruction execution. While CPUs typically employ complex out-of-order execution and branch prediction mechanisms to optimize single-threaded performance, GPUs use a simpler in-order execution model called Single Instruction Multiple Threads (SIMT). In SIMT, groups of threads, called warps, execute the same instruction simultaneously, which simplifies the control logic and allows for greater parallelism. However, this also means that divergent control flow within a warp can lead to performance degradation, as different execution paths must be serialized.

In summary, GPUs have revolutionized the field of computer vision and deep learning by providing a highly parallel architecture and powerful computational capabilities. Their numerous simpler cores, specialized memory types, and throughput-oriented design make them more suitable for processing large-scale, high-dimensional data than traditional CPUs. Understanding the architectural differences between GPUs and CPUs, as well as the intricacies of GPU memory management and execution models, is essential for developing high-performance computer vision and deep learning applications.

2.2.2 CUDA Platform

The CUDA (Compute Unified Device Architecture) platform, developed by NVIDIA[23], is a parallel computing platform and programming model that allows developers to harness the power of NVIDIA GPUs for general-purpose computing tasks [90]. CUDA has had a significant impact on various fields, including scientific computing, machine learning, and computer vision, by enabling researchers and engineers to leverage GPU acceleration for their applications [94, 66].

CUDA provides a C/C++ programming interface, along with libraries and tools, for writing parallel algorithms that can be executed on the GPU [23]. The CUDA programming model is based on the concept of threads, blocks, and grids. Threads are the smallest execution units and are grouped into blocks, which are further organized into a grid [90]. Each thread in a block can access a shared memory space, enabling efficient communication and data sharing between threads within the same block. The grid is a higher-level abstraction that represents the entire problem domain and is divided into blocks.

In the CUDA programming model, the GPU is treated as a highly parallel co-processor that works in conjunction with the CPU [90]. The CPU, referred to as the host, is responsible for executing the main program, managing memory transfers between the host and the device (GPU), and launching kernels on the device. Kernels are functions that are executed on the GPU by a large number of threads in parallel [90].

The CUDA platform exposes the GPU's hierarchical memory architecture, which includes global memory, shared memory, constant memory, and texture memory [23]. Global memory is the largest and slowest memory space, accessible by all threads in the grid, and can be used to store large data structures. Shared memory, which is faster than global memory, is shared among threads within the same block and can be used for inter-thread communication and data sharing [90]. Constant memory is a read-only memory space that can be used to store constant data used by all threads, while texture memory is a cached memory space optimized for 2D and 3D data access patterns [23].

To optimize the performance of CUDA applications, developers need to carefully manage the memory hierarchy and thread execution [109]. For example, they should minimize global memory access, use shared memory and constant memory to reduce memory latency, and design the kernel to maximize thread-level parallelism and occupancy [126].

CUDA provides a rich set of libraries, such as cuBLAS, and cuDNN [20], which offer GPU-accelerated implementations of commonly used functions in

linear algebra, signal processing, sparse matrix operations, and deep learning. These libraries allow developers to easily integrate GPU acceleration into their applications without having to write custom GPU kernels [94].

The CUDA platform also includes various tools for profiling and debugging, such as the NVIDIA Visual Profiler [91]. These tools enable developers to analyze the performance of their GPU-accelerated applications, identify bottlenecks, and debug their code. They can also help in optimizing memory usage, identifying performance-limiting factors, and ensuring the correctness of the code [23].

To sum up, the CUDA platform has played a crucial role in democratizing GPU computing and has made it accessible to a broad range of developers and researchers. Its programming model, rich set of libraries, and development tools have enabled the rapid development and deployment of GPU-accelerated applications in various fields, including computer vision, deep learning, and scientific computing [94, 66, 23]. By leveraging the powerful capabilities of NVIDIA GPUs, the CUDA platform has significantly contributed to the advancement of these fields and has set the stage for further breakthroughs and innovations.

cuBLAS

NVIDIA's cuBLAS [91] is a GPU-accelerated library that provides implementations for a wide range of Basic Linear Algebra Subprograms (BLAS). It is an essential component in the matrix multiplication operation, a core computation in deep learning models. By leveraging the vast parallel computing capability inherent to GPU architectures, cuBLAS offers highly optimized routines that significantly enhance the performance of matrix multiplication. The strength of cuBLAS lies in its meticulous design to exploit the capabilities of NVIDIA GPUs. The library includes different versions of matrix multiplication routines tailored for various scenarios, such as small or large matrices, dense or sparse matrices, single or double-precision computations, and more. These routines are highly parameterizable, offering flexibility in how they're used based on the specific requirements of the operation at hand.

The influence of cuBLAS extends to major deep learning frameworks such as TensorFlow and PyTorch, both of which utilize cuBLAS as their backend for executing matrix multiplication operations on NVIDIA GPUs. By doing so, these frameworks can deliver high-performance computation, leading to faster training and inference times for deep learning models. The extensive use of cuBLAS in these frameworks underscores its effectiveness in accelerating matrix multiplication and, in turn, the overall performance of deep learning models. However, as with any library, the performance of

cuBLAS is contingent upon the GPU architecture it's used with, meaning that the specific GPU's features and specifications can influence the acceleration achieved.

Tensor Cores

Tensor Cores [91] are innovative hardware units developed by NVIDIA and integrated into their Volta, Turing, and subsequent architectures. These units have been explicitly designed to accelerate matrix arithmetic, which is essential for deep learning operations. They allow significantly faster training and inference of neural networks by performing mixed-precision matrix multiply-and-accumulate calculations in a single operation. In more detail, a Tensor Core takes as input two half-precision floating-point matrices (FP16) and one full-precision matrix (FP32), performs a matrix multiplication of the half-precision matrices, and then adds the full-precision matrix to the result. The operation can be written as $D = A * B + C$, where A and B are FP16 matrices, C and D are FP32 matrices, " $*$ " denotes matrix multiplication, and " $+$ " signifies matrix addition. This operation is common in deep learning computations, particularly in convolution and transformer-based architectures.

The ability of Tensor Cores to perform this operation in a single step leads to dramatic speedup in computation. They can deliver up to 125 teraFLOPS of performance for mixed-precision matrix multiplication in a single GPU, making them instrumental in achieving high performance in deep learning tasks. Moreover, using mixed-precision arithmetic, where the bulk of the computation is performed in lower precision, can lead to further acceleration and memory savings without significant loss in model accuracy. NVIDIA provides software support for using Tensor Cores in popular deep learning frameworks, making their power accessible to the deep learning community. Figure 2.8 illustrates the usage of Tensor Cores in a matrix multiplication operation.

2.2.3 High-Performance Computing with GPUs

High-performance computing (HPC) stands at the forefront of computational advancement, enabling researchers and industries to tackle complex, data-intensive tasks with unprecedented efficiency and speed. Over the past decade, Graphics Processing Units (GPUs) have emerged as a pivotal tool within the HPC landscape[93].

GPUs, originally designed for rendering images in computer graphics, have expanded their influence far beyond their initial purpose. Their ar-

Figure 2.8: Matrix Multiplication Operation with Tensor Cores[91]

chitecture, comprising hundreds to thousands of cores, allows for efficient parallel processing of computational tasks. This feature is well-aligned with the needs of HPC, where the simultaneous execution of tasks can significantly reduce computational times and improve the handling of large-scale problems.

The GPU's ability to accelerate computational workloads has found wide application in areas such as scientific computing, deep learning, and big data analytics. Compared to traditional Central Processing Units (CPUs), GPUs offer superior throughput and energy efficiency, making them particularly suitable for data-intensive computations in HPC.

Another significant aspect of GPUs in HPC is their compatibility with programming models and languages that have been widely adopted by the scientific community. For instance, NVIDIA's CUDA (Compute Unified Device Architecture) platform[23] provides a software environment that allows developers to use the C programming language to code algorithms directly into GPU instructions. This has greatly eased the implementation of GPU-based computation in HPC. The integration of GPUs into HPC systems has substantially enhanced the capability to handle 'big data.' As the volume of data generated by various fields continues to explode, the need for computational platforms that can quickly process and analyze this data becomes critical. GPUs, with their high-performance computing capabilities, provide a solution to this challenge.

Moreover, the advent of GPU-accelerated supercomputers has set new milestones in HPC. Notably, systems like the Summit at Oak Ridge National Laboratory in the United States and TSUBAME at the Tokyo Institute of Technology in Japan represent this major shift. The Summit supercomputer leverages thousands of GPUs, delivering unprecedented computational capabilities and setting the pace for scientific discovery. Similarly, the TSUBAME series, known for its GPU-accelerated computing power, have been instrumental in pushing the boundaries of HPC. The latest in this series, TSUB-

AME 3.0[112], has garnered international attention for its efficient power usage and significant computational capacity, showcasing the efficacy of GPU utilization in HPC.

These innovative supercomputers underline the critical role GPUs play in modern HPC systems and highlight a significant shift in supercomputer design and utilization. The ability to harness the power of GPUs at this scale opens up unprecedented possibilities for tackling large-scale, complex problems across a multitude of research domains.

The future of HPC with GPUs appears bright, as ongoing advancements promise to elevate their role further. Emerging trends like exascale computing and artificial intelligence, heavily reliant on GPU-accelerated computation, are pushing the frontiers of what is possible in HPC. These developments suggest a continual and integral role for GPUs in the evolution of high-performance computing.

Chapter 3

Speed-up Single Shot Detector on GPU with CUDA

3.1 Introduction

Object detection is a vital component of computer vision that aims to recognize and locate real-world objects in images or video sequences. Humans possess an innate ability to recognize objects regardless of changes in viewpoint, scale, translation, or rotation. Even when an object is partially obscured, humans can often identify it with ease. The ultimate goal in the field of computer vision is to enable contemporary computing systems to detect objects with human-like precision while maintaining the high-speed processing capabilities of computers. Despite significant advancements in recent years, this problem remains a challenging task.

Artificial Neural Networks (ANNs), inspired by the biological nervous systems, have been proposed as a means to teach computers to learn in a manner similar to humans. Artificial neural networks are composed of individual "neurons" with weights that mimic the connections found in biological neural networks. The process of determining each neuron's weight is referred to as training, while inference involves using a trained neural network to perform a computing task. An ANN consists of at least two layers, input and output, each containing numerous perceptrons. The "hidden layer" is the layer that sits between the input and output layers. A deep neural network is an ANN with multiple hidden layers.

As object detection technology has rapidly evolved, its commercial value has been recognized in various fields, such as large-scale security surveillance and autonomous driving. Numerous commercial companies have launched products or services based on object detection, leading to a continuously

growing market. However, an efficient and reliable object detection system that meets commercial demands has yet to be developed. For commercial applications, detection speed remains a critical challenge [56]. While object detection using deep convolutional neural networks has made significant progress in recent years, achieving real-time detection without sacrificing accuracy is still an ongoing pursuit.

Object detection involves identifying the presence of an object of interest in an image or video frame, regardless of its size, orientation, or environment. Various methods exist for performing detection tasks, with the simplest being the template matching technique using a moving window slide, such as in R-CNN [42]. However, the computational time required by this approach is a significant bottleneck.

In this research, we explore techniques for accelerating object detection algorithms, specifically focusing on deep learning-based methods, to strike a balance between accuracy and speed. We aim to develop a real-time object detection system that maintains high accuracy while addressing the computational challenges associated with deep learning. By leveraging the power of GPUs and the CUDA platform, we aim to optimize the performance of object detection algorithms for commercial applications, enabling the deployment of efficient and reliable systems in various industries.

The increasing resolutions of images and videos present a growing challenge for object detection tasks. Sequential processing of high-resolution images and videos using a single-core processor can no longer satisfy the required speedup for detecting objects. Consequently, contemporary research has shifted its focus to parallel processing, which involves performing multiple calculations simultaneously using either a hardware paradigm, such as GPUs, or a software-hardware paradigm, such as FPGAs. Both paradigms offer superior performance compared to single-core CPUs.

GPUs, initially designed for rendering graphics, have evolved to serve as general-purpose accelerators. NVIDIA's Compute Unified Device Architecture (CUDA) technology is now frequently employed to accelerate image and video processing applications beyond the realm of graphics. In 2010, S. Mehta et al. presented a high-performance implementation of the Sum of Absolute Differences (SAD) [87], a measure of similarity between image blocks, on a general-purpose GPU architecture using NVIDIA's CUDA. Their research demonstrated that object detection could achieve faster detection speeds through GPU acceleration while maintaining satisfactory results. The recent advancements in GPU computing capabilities have paved the way for the real-time execution of highly complex computer vision algorithms.

Multithreaded data-parallel GPU architectures have found widespread applications in fields such as advanced driver assistance systems (ADAS),

autonomous driving, scene interpretation, intelligent video analytics, and facial recognition, among others. The potential for further advancements and applications of GPU-accelerated object detection is immense, making it an exciting and promising area of research.

In this work, we explore the possibilities of harnessing the power of GPUs and CUDA technology to further improve the speed and efficiency of object detection algorithms. By optimizing the parallel processing capabilities of GPUs and developing innovative techniques to accelerate deep learning-based object detection methods, we aim to create a robust and real-time object detection system suitable for various commercial and industrial applications.

One of the most widely used detection networks is the Single Shot Multibox Detector (SSD) [78]. It achieves a high mean Average Precision (mAP) compared to other high-speed detection networks, striking a favorable balance between accuracy and detection speed. As a result, SSD serves as an excellent subject for research in object detection acceleration.

Most object detection networks are built upon convolutional neural networks (CNNs), and numerous techniques have been developed to speed up the computation of CNN's convolution and fully connected layers. However, in detection networks, other modules besides convolution and fully connected layers also require a significant amount of processing time. Therefore, in addition to optimizing the CNN layers, attention should be given to the calculation of other layers. By accelerating these layers using GPUs, we can achieve faster detection speeds, which is a primary goal of our research.

Our main contribution is the acceleration of the SSD512 detection through the use of CUDA, resulting in a performance improvement of approximately 22.53% compared to the standard version [73], a trade-off between speed and accuracy that we do extremely well. We implemented all components using CUDA [111] and C++, and rewrote the Backbone (VGG16) with cuDNN, which made the backbone about 9.19% faster than the original version. Leveraging the GPU's capabilities, we processed the image and classification tasks in parallel. Finally, we proposed a CUDA-based Non-maximum Suppression (NMS) algorithm that is approximately 52.61% faster than the original version. This research demonstrates the potential for further improvements in object detection speed and efficiency by optimizing both CNN layers and other modules using GPU acceleration and CUDA technology.

3.2 Related Work

In this section, we highlight and discuss the research efforts that bear significance and relevance to our own investigation.

3.2.1 Object detection

Since 2010, manual feature extraction-based object detection algorithms faced limitations in their development. Classical techniques consist of three steps: region selection, feature extraction, and classification. However, these techniques encountered two significant issues: the absence of an efficient region-selection method with a simple time complexity and the lack of robustness in manual feature extraction. In object detection competitions such as the "ImageNet Large Scale Visual Recognition Challenge (ILSVRC)" [108], "COCO: Common Objects in Context Detection Challenge" [77], and "PASCAL VOC: The PASCAL Visual Object Classes Challenge" [32], deep neural network-based methods consistently outperformed other approaches.

In 2012, a breakthrough method based on convolutional neural networks (CNN) was introduced by Krizhevsky et al. [66], revolutionizing the object detection landscape. CNNs significantly improved both region selection and feature extraction, particularly in region proposal-based approaches like R-CNN [42].

Modern object detection methods can be broadly categorized into two groups: one-stage frameworks and two-stage frameworks. One-stage frameworks include YOLO (You Only Look Once) [103], SSD (Single Shot Multi-Box Detector) [78], and RetinaNet [76], while two-stage frameworks comprise Faster R-CNN [105], Mask R-CNN [46], and Cascade R-CNN [14].

- **Two-stage frameworks**: Pioneered by the introduction of the Region-CNN (R-CNN)[42] and its successors, Fast R-CNN[41] and Faster R-CNN[105], these frameworks primarily focus on accuracy. The two-stage approach starts with the first stage which generates a number of region proposals which are potential areas in the image where the object could be. These region proposals are not actual bounding boxes, but areas where the algorithm believes an object is present based on the features learned. The second stage uses these region proposals to not only refine the bounding box coordinates, i.e., improve the localization of the object, but also to predict the classification scores for each refined box. While these methods offer high accuracy, the process is inherently slow due to the two-stage structure and the complexity of the operations, making real-time applications a challenge.

- **One-stage frameworks**: To overcome the speed limitations of two-stage frameworks, one-stage frameworks were introduced. These methods, like YOLO (You Only Look Once)[103] and SSD (Single Shot Multibox Detector)[78], aim for faster detection speeds by merging the

two stages into a single network pass. In one swift operation, they predict bounding box coordinates and classification results simultaneously. These methods eliminate the need for region proposal generation, thus substantially increasing the speed of object detection. However, while their speed is commendable, they often lag behind in terms of accuracy compared to their two-stage counterparts. This is primarily because they have to balance speed and accuracy, and the single pass through the network doesn't always allow for the careful refinement of bounding boxes that two-stage methods provide.

It's important to note that the choice between one-stage and two-stage frameworks often depends on the specific requirements of a task. If accuracy is paramount and processing time is less of an issue, a two-stage method might be preferred. Conversely, if real-time detection is needed, a one-stage method could be a better choice. Our work, in this case, is focused on improving the speed of one-stage detectors, specifically SSD, without significantly compromising accuracy.

The imbalance between background and non-background regions is the primary reason for these differences. In two-stage frameworks, background samples are filtered through the region proposal stage, reducing the number of candidates. One-stage frameworks, on the other hand, are more prone to false positives due to the larger number of regions classified as background. To address this issue, RetinaNet introduced a new loss function called Focal Loss, which helps maintain a balance between background and non-background regions during training.

In two-stage frameworks, a region proposal network (RPN) generates possible bounding boxes, which are then refined by the detection head network in terms of bounding box coordinates and classification scores. In contrast, one-stage frameworks predict box coordinates and classification results in a single network pass. Non-maximum suppression (NMS) serves as a crucial post-processing step, removing boxes with similar locations and shapes but with lower confidence levels. Huang et al. [56] provided a comprehensive analysis of the performance of various object detection frameworks, examining not only the accuracy of different approaches but also their detection speeds, which is an important indicator of a framework's maturity.

In conclusion, over the past ten years, there has been a significant evolution in the field of object detection approaches. Accuracy and computing efficiency have improved because of Convolutional Neural Networks (CNNs) and other deep learning approaches. The choice between a one-stage and two-stage framework depends on the application's details as well as the balance between detection speed and accuracy that is deemed appropriate for

the particular situation. These frameworks offer a wide range of possibilities for practitioners in a variety of domains, helping to push the limits of what is possible for object identification jobs even though they are not without trade-offs.

3.2.2 High Speed of Feature extraction

In the realm of feature extraction, speed is of the essence. The faster the algorithm, the more efficient the overall process becomes, leading to improved performance and productivity. Two algorithms that stand out in this regard are Winograd's minimal filtering algorithm [68] and the Fast Fourier Transform (FFT)[24].

Winograd's Minimal Filtering Algorithm

Winograd's minimal filtering algorithm is an efficient method for convolution, a fundamental operation in feature extraction. Named after Shmuel Winograd, this algorithm reduces the number of multiplications in the convolution process, thereby increasing the speed of computation. The algorithm works by transforming the convolution operation into a set of smaller, simpler operations that can be computed more quickly. It does this by exploiting the redundancy in the computation of overlapping output values. The transformed operations are then computed using a minimal number of multiplications, hence the name "minimal filtering algorithm". Winograd's minimal filtering algorithm is particularly effective in the context of convolutional neural networks (CNNs), where convolution operations are a key component. By reducing the computational complexity of these operations, the algorithm significantly speeds up the feature extraction process in CNNs.

Fast Fourier Transform (FFT)

The Fast Fourier Transform (FFT) is another algorithm that plays a crucial role in feature extraction. FFT is an algorithm to compute the Discrete Fourier Transform (DFT) and its inverse, which are mathematical techniques used to transform a function into its constituent frequencies. FFT reduces the computational complexity of calculating the DFT from $O(n^2)$ to $O(nlogn)$, where n is the data size. This makes it an extremely efficient method for transforming time-domain signals into frequency-domain signals, a common operation in feature extraction. In the context of image processing, for example, FFT can be used to identify the frequency components of an image, which can then be used as features for tasks such as image recognition or

classification. The high speed of the FFT algorithm makes it a valuable tool for feature extraction in large-scale data analysis.

Both Winograd's minimal filtering algorithm and the Fast Fourier Transform contribute significantly to the high speed of feature extraction. Their efficiency and speed make them indispensable tools in the field of data analysis and machine learning.

3.2.3 Single Shot Multibox Detector (SSD)

The Single Shot Multibox Detector (SSD) is a widely-used one-stage framework that improves upon the limitations of the R-CNN family of algorithms. In contrast to R-CNN, which requires multiple iterations to compute region proposals and classifications, SSD combines these tasks into a single pass through the CNN. As the name "Single Shot" implies, this streamlined approach accelerates the object detection process.

SSD Architecture

The SSD architecture is designed to output multi-scale bounding boxes from multiple output layers, enabling it to detect objects of various sizes effectively. The model architecture, as illustrated in Figure 3.1, is based on the VGG16 network [115], which serves as the backbone for feature extraction.

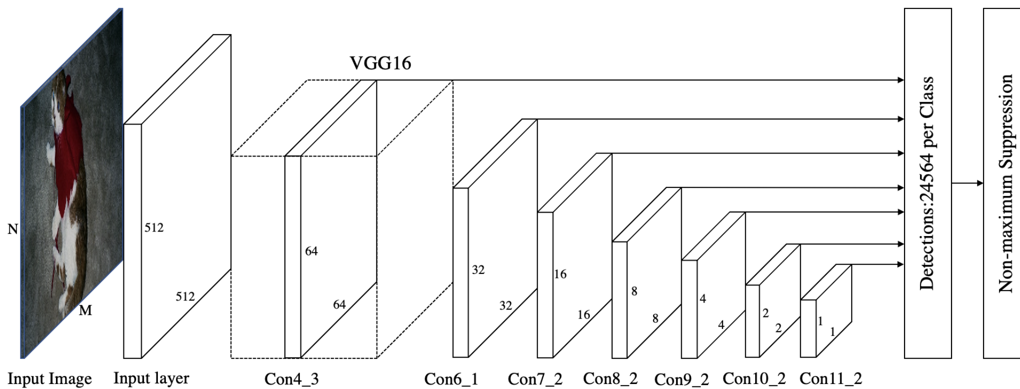

Figure 3.1: Single Shot Multibox Detector Model (SSD512)

Depending on the input image size, SSD is mainly divided into two variants: SSD300 [78] and SSD512, as shown in Figure 3.1. The first part of the network, known as the base network, follows a standard architecture for

image classification and is responsible for feature extraction. The following layers serve as auxiliary structures for detecting multiple objects using multi-scale feature maps.

Multi-Scale Feature Maps

Multi-Scale Feature Maps capture information at different spatial scales in an image, accommodating objects and patterns of varying sizes. There are several ways to implement multi-scale feature maps in Convolutional Neural Networks (CNNs)[58]. One approach is the use of pyramidal architectures, where the input image is processed at different resolutions. Each resolution captures features at a distinct scale, which are then combined to provide a multi-scale feature representation.

Another approach is through Inception modules, introduced in GoogLeNet [120]. These modules perform convolutions and pooling operations at multiple scales in parallel, then concatenate the results. This allows the network to learn features at different scales simultaneously. Dilated (atrous) convolutions [133] offer yet another technique. They introduce a dilation factor to standard convolutions, expanding the receptive field without increasing the number of parameters, allowing the model to capture larger scale features. Lastly, Feature Pyramid Networks (FPNs) [75] construct a pyramid of feature maps at different scales, combining low-level, high-resolution features with high-level, low-resolution features. This combination offers a rich multi-scale feature representation. These methods enhance a CNN's ability to recognize objects and patterns of varying sizes, improving its performance in many computer vision tasks.

Non-maximum Suppression

Non-Maximum Suppression (NMS) is a technique employed in various computer vision tasks. It is a class of algorithms designed to select one entity, such as bounding boxes, out of many overlapping entities. The selection criteria can be customized to achieve the desired results, typically involving some form of probability measure and an overlap metric like Intersection over Union (IoU). NMS plays a crucial role in high-resolution image detection, where more candidate regions are generated, increasing the NMS computation time. In all object detection frameworks, NMS is an essential algorithm that influences detection accuracy and speed.

To enhance detection accuracy, several NMS variants have been proposed [10, 52, 49]. Soft NMS [10] reduces the confidence score as a continuous function of IoU and retains all boxes, rather than discarding them. Continuous

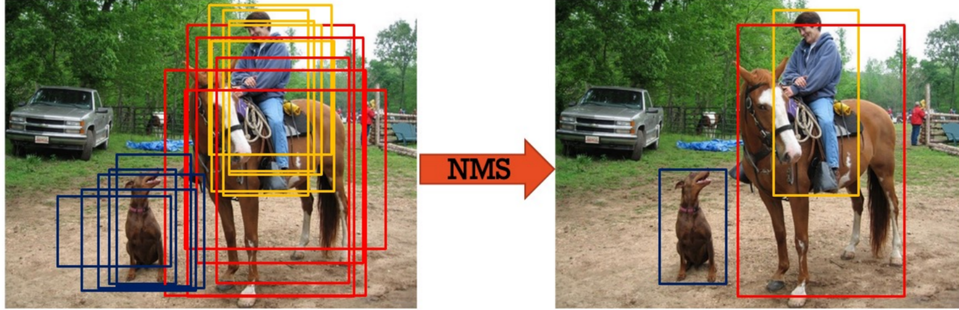

Figure 3.2: Non-maximum Suppression algorithm

functions are manually designed in [10], and a specialized network is learned in [52] to regain confidence. Additionally, [49] improves localization accuracy by updating bounding box coordinates when suppressing adjacent boxes, in addition to changing the confidence.

In terms of time cost, the worst-case complexity for NMS is $O(n^2)$, where N is the number of boxes. As a result, the time cost becomes high when the number of boxes is large. This issue is particularly pronounced in crowded scene object detection, where thousands of boxes are generated by the RPN of Faster R-CNN, and the number of candidate boxes can even exceed 10,000, especially as high-resolution image detection gains popularity.

The map-reduce approach presented in [92] serves as the foundation for our implementation, which includes several enhancements. Firstly, the method in [92] requires sorted input, while our approach accepts inputs in any order. Secondly, [92] does not account for the non-transitive property.

We start with the method described in [92], as shown in Figure 3.3[92]. Let each object class have a bit matrix M of size $N \times N$. The matrix is defined as follows:

$$M_{i,j} = \begin{cases} 1 & \text{if} \quad i > j \quad \text{and} \quad IOU_{i,j} > threashold \\ 0 & \text{otherwise} \end{cases}$$

$M_{i,j}$ is a symmetric matrix, as shown in Figure 3.3, since $IOU_{(i,j)} = IOU_{(j,i)}$. All of the 1 components in the lower triangle are marked as -1. This is the stage of the map. Row by row, the matrix is reduced: if a row of the matrix has a -1 element, output 1, else output 0.

Figure 3.3 presents a simplified illustration of the proposed algorithm. In the given image frame, they have three objects, three window clusters, and

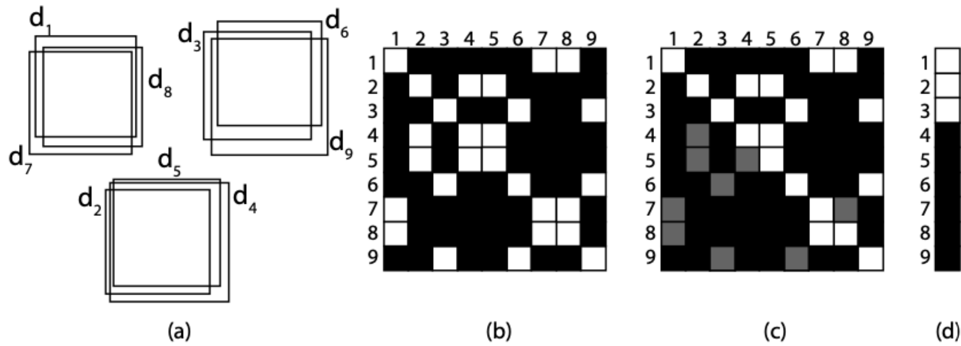

Figure 3.3: Visualization of Map-reduce NMS proposal[92]

nine detections. Their matrix aims to encode the relationship between all detections, with the initial assumption that all are potential cluster representatives (indicated by a matrix filled with ones).

The algorithm proceeds as follows:

1. First, they decide if two candidate areas d_i and d_j belong to the same cluster based on whether their areas overlap beyond a given threshold. If they do not, we assign a zero to the corresponding matrix coordinates (d_i, d_j) and (d_j, d_i).

2. Next, they evaluate the non-zero values in each row. They then assign a zero if the area of the detection indexed by the row (d_i) is strictly smaller than that of the detection indexed by the column (d_j). As a result, d_i is discarded as a potential cluster representative.

3. Finally, a horizontal AND reduction operation is performed. This operation preserves a single representative per cluster, thereby completing the Non-Maximum Suppression (NMS) process.

Through these steps, their algorithm effectively carries out the task of object detection and selection within the given image frame. The GPU-based algorithm exhibits varying acceleration effects depending on the number of boxes. One possible enhancement is to optimize the computation of the usage of shared memory, exploiting the parallel processing capabilities of the GPU. Additionally, they can explore more efficient sorting and data organization methods to further speed up the NMS process. Another aspect to consider

is the adaptability of the algorithm to different object detection frameworks, ensuring that the improved NMS method can be seamlessly integrated into various models.

In conclusion, our research aims to build upon the existing GPU-based NMS method to create a more efficient and adaptable algorithm for object detection tasks. By implementing enhancements to the computation and sorting, we hope to create an improved NMS algorithm that can be seamlessly integrated into a wide range of object detection frameworks and deliver superior performance than previous work.

3.3 Our Implementation and Optimization

In this section, we will first discuss how SSD512 is constructed using CUDA and how to implement and improve the NMS algorithm using CUDA. Detailed results are discussed in the next chapter.

3.3.1 Problem of Current Research

In SSD512[78], we divide it into four parts: pre-processing layer, feature extraction layer, proposal layer, and post-processing layer. We use [73] as the test object. In our test, the most time-consuming part is the feature extraction layer, which accounts for 55.38% of the total computation time. However, the remaining three parts also account for nearly half of the overall processing time. In the original SSD512 implementation, only the feature extraction layer is calculated on the GPU. The remaining three parts are all computed on the CPU. Therefore, we propose to move these three parts to the GPU to accelerate the computation. Later, we will introduce our GPU-based implementation method.

3.3.2 Pre-Processing

During the preprocessing phase of our approach, we focus on utilizing the GPU efficiently to carry out tasks that prepare the input images for subsequent processing. These tasks include resizing the input images to a dimension suitable for the network and normalizing the images by subtracting the mean RGB values. Normalization assists in mitigating the effects of lighting conditions and other environmental factors on object detection, thereby enhancing the model's robustness. To streamline this process, we assign a thread to each output pixel in the resized image, allowing for concurrent processing of each output pixel. This concurrent processing strategy enhances

the performance of the preprocessing stage.

The preprocessing tasks are executed within a single GPU function, referred to as a CUDA kernel. This approach helps to minimize the overhead associated with launching multiple kernels and transferring data between the CPU and GPU. Efficient memory management is also an integral part of our optimization strategy. Techniques such as using shared memory for rapid data access and minimizing global memory accesses are investigated. These optimization strategies are applied to various operations within the preprocessing stage, including resizing and normalization, with the aim of further enhancing performance. The performance results affirm the effectiveness of these approaches.

3.3.3 Feature Extraction Layer

In the feature extraction stage, we employed the cuDNN library in our implementation. This choice was made due to cuDNN's highly optimized routines for deep neural networks, which can significantly boost the performance of our model. In particular, our implementation focuses on enhancing the efficiency of the VGG16 network, a critical component in the SSD model for feature extraction. The original PyTorch version of SSD also utilizes Python, a high-level interpreted language. While Python is known for its easy-to-use syntax and rich ecosystem, it does not always provide the most efficient execution for low-level, compute-intensive operations like convolution, which is widely used in VGG16. PyTorch already leverages cuDNN under the hood for many of its operations, including those used in the VGG16 feature extractor. However, by explicitly rewriting this layer with cuDNN, we are able to ensure relatively more optimal use of cuDNN's capabilities according to different CNN models, which may not always be achieved with PyTorch's high-level interface. By contrast, our cuDNN-accelerated version is implemented in C++, a lower-level language that allows for more fine-grained control of the computing resources. With cuDNN, we can directly tap into the optimized routines for convolution, pooling, and other operations, provided by NVIDIA. The library is specifically designed to leverage the GPU's capabilities to their full extent, resulting in faster execution times for feature extraction.

Additionally, cuDNN provides automatic tuning functionality that can select the optimal algorithm settings based on the specific GPU architecture and the size of the input data. This feature further contributes to the improved efficiency of our feature extraction stage. It's essential to mention that these performance gains are hardware-dependent and can vary based on factors like the specific GPU model used, its memory bandwidth, and the

size of the dataset processed. Despite these variables, the use of cuDNN and C++ for feature extraction generally leads to a more efficient computation compared to the original Python-based PyTorch version.

3.3.4 Proposal Layer

In the proposal layer stage of our implementation, the aim is to generate a set of potential bounding boxes or 'anchors' within the input images that likely encapsulate objects of interest. This process is crucial for subsequent object detection layers within the network. The proposal layer is optimized using GPU-accelerated processes. Specifically, a unique GPU thread is assigned to each element, where an element could be an anchor or a candidate region. This one-to-one mapping ensures concurrent processing of each element, which substantially improves the efficiency of the proposal layer stage. Moreover, in the pursuit of optimization, efforts are made to condense the processing into as few kernels as possible. This strategy helps to reduce the overhead associated with launching multiple kernels and minimizes data transfer between the CPU and GPU, thereby enhancing the efficiency of the overall process.

In the generation of proposals, strategies have been employed to create anchors that consider varying aspect ratios and scales of potential objects within the input images. This ensures the creation of bounding boxes suitable for the detection of objects with diverse sizes and shapes. The proposal layer also includes the Softmax function, an essential component that normalizes the confidence scores associated with each proposed region. The implementation of Softmax on the GPU, as shown in algorithm 1, has provided acceleration in the processing of the proposal layer.

Moreover, we adopted various memory management and data processing techniques to further optimize the proposal layer's performance. Utilizing shared memory for quicker data access, minimizing global memory accesses, and optimizing parallel reduction operations are some of the key steps taken. It is crucial to note that these enhancements not only improved the efficiency of individual components within the SSD512 object detection framework but also increased the overall performance of the system. However, performance can still vary depending on factors such as the specific GPU model used, its memory bandwidth, the size of the dataset processed, and the complexity of the images.

Algorithm 1: Softmax with GPU

Input: fea_in, class_num, height, channel
Output: fea_out

1 $i \leftarrow threadIdx.x + blockIdx.x * blockDim.x$;
2 $j \leftarrow threadIdx.y + blockIdx.y * blockDim.y$;
3 $k \leftarrow blockIdx.z * class_num$;
4 **if** $k < channel$ **then**
5     $k_iter \leftarrow k$; $step \leftarrow height * channel$
6   **for** $k_iter < (k + class_num)$; **do**
7         $coef += expf(fea_in[i * step + j * channel + k_iter])$
8   **end**
9     $coef \leftarrow 1/coef$;
10   **for** $k_iter < (k + class_num)$; **do**
11         $fea_in[i * step + j * channel + k_iter] \leftarrow expf(fea[i * step + j * channel + k_iter]) * coef$;
12   **end**
13 **end**
14 $fea_out \leftarrow fea_in$;

3.3.5 Post Processing

The post-processing stage of the implementation, particularly the Non-Maximum Suppression (NMS) algorithm, has been an area of focus for optimization efforts. Non-Maximum Suppression is a common technique in object detection, used to eliminate overlapping bounding boxes, ensuring that each object is only detected once. The use of shared memory and algorithmic adjustments to cater to object detection tasks has resulted in improvements in processing times. The NMS algorithm plays a crucial role in object detection tasks. Its primary function is to refine the candidate object detections by selecting the most representative bounding box from each cluster of similar detections. The process of NMS can be likened to a clustering problem involving two essential operations: (1) classifying each detection into a particular cluster, and (2) selecting a representative for each cluster.

To maximize the inherent parallelism offered by general-purpose GPUs as much as possible, the NMS kernel is designed such that each processing thread can independently assess the overlap between two specific bounding boxes. The goal is to minimize data dependencies that could potentially enforce serialized computations, thereby addressing the scalability challenges associated with the conventional iterative clustering procedure. It's important to note that while improvements have been achieved in our method,

about 52.61% faster than the original version, the effectiveness of these optimizations can still be influenced by factors such as the specific GPU architecture, different sizes of shared memory, the complexity of the images processed, and the characteristics of the bounding box clusters.

Algorithm 2: Our NMS method with GPU

```
1  Function nms_kernel(d_bboxes, d_nms, num_bboxes, threshold):
2      shared_coords, shared_scores ← allocate shared memory
3      block ← this thread block
4      idx ← get current thread index
5      if idx < num_bboxes then
6          my_bbox ← d_bboxes[idx]
7          store my_bbox into shared_coords and my_bbox.score into
             shared_scores at position of threadIdx.x
8          sync block
9          for i ← threadIdx.x+1 to min(blockDim.x, num_bboxes) do
10             other_bbox ← get from shared_coords at position of i
11             iou ← IoU(my_bbox, other_bbox)
12             mask ← parallel comparison(iou > threshold)
13             if threadIdx.x is a multiple of 32 then
14                 d_nms[i + blockIdx.x * blockDim.x] ← (mask == 0)
15             end
16             sync block
17         end
18     end
```

This kernel function `nms_kernel` takes algorithm 2 in an array of BoundingBox structures, an array to store the NMS results, the total number of bounding boxes, and an overlap threshold as its parameters. Each BoundingBox structure represents a detected object and includes its bounding boxes coordinates (x, y, w, h) and detection score. The `cooperative_groups` namespace is part of CUDA's Cooperative Groups programming model. In this code, it is used to create a thread block, which represents a group of threads that can synchronize and share resources. The `shared_mem` shared memory space is declared to store the coordinates and scores of each bounding box in the thread block. Shared memory is a type of on-chip memory that allows threads within the same block to share data.

The kernel function then runs a loop over all bounding boxes. For each bounding box, it calculates the Intersection-over-Union (IoU) with all other bounding boxes. If the IoU exceeds the threshold, it means the bounding

boxes are overlapping relatively significantly and only the one with the higher score should be kept. The `ballot_sync` function is a warp-level primitive that collects the predicate value from all active threads in the warp and returns a mask. If the IoU is greater than the threshold, the corresponding bit in the mask is set to 1, indicating that the bounding box should be suppressed. Finally, the kernel writes the suppression result back to the `d_nms` array in global memory. The suppression result is represented as a binary mask, where 1 means the bounding box should be kept and 0 means it should be suppressed.

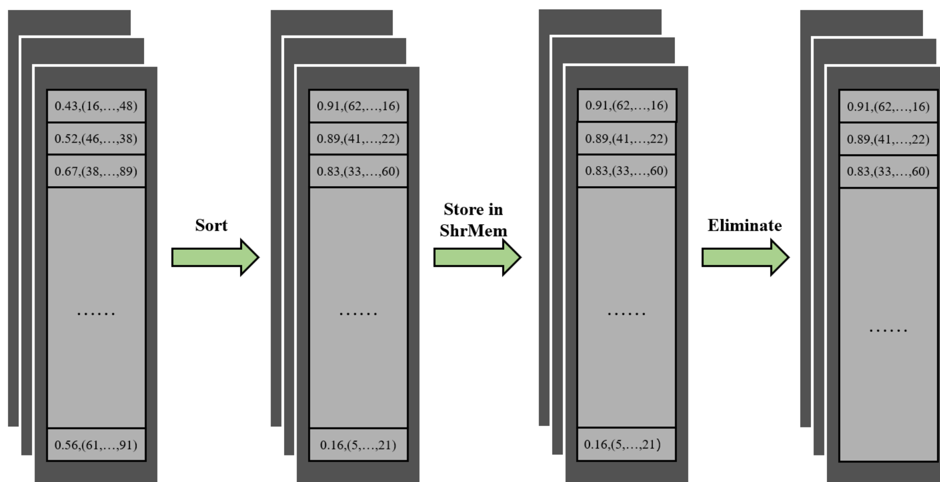

Figure 3.4: Post Processing Data Flow

The post-processing data flow in our method consists of multiple stages, as illustrated in Figure 3.4. Each dark grey box in this figure represents a CUDA block, and each light grey box symbolizes a CUDA thread. The numerical values within each thread correspond to score-coordinate pairs for the default bounding boxes.

In the initial stage, all bounding boxes are sorted based on their scores. Sorting is an important step as it determines the order in which the bounding boxes are evaluated and eliminated. Higher-scored bounding boxes are considered more likely to contain an object and are therefore processed first. After sorting, the score-coordinate pairs are stored in the shared memory for efficient access. Shared memory has lower latency and higher bandwidth than global memory, making it suitable for storing data that needs to be frequently accessed. In the next stage, each thread retrieves the bounding

box with the highest score from the shared memory. This is the "target" bounding box for that thread. Subsequently, each thread calculates the Intersection over Union (IoU) between its target bounding box and all other bounding boxes. The IoU is a measure of the overlap between two bounding boxes. If the IoU of a bounding box with the target bounding box exceeds a certain threshold, it is considered to be part of the same object detection cluster and is eliminated.

Through this data flow, the proposed method maximizes parallelism as much as possible by ensuring that each thread independently handles a different bounding box. Furthermore, it leverages shared memory to reduce memory access latency, contributing to an overall acceleration in the post-processing stage.

The use of Algorithm 2 necessitates the sorting of bounding boxes according to their scores. We implemented the sorting algorithm on the GPU, adapting the merge sort algorithm for parallel processing [34]. Additionally, we pruned boxes with lower scores, given that the quantity of remaining boxes impacts detection accuracy.

Our parallel merge sort comprises three stages. Initially, we divide the input data into n equal-sized segments. Following that, we employ n thread blocks to sort each of these segments. Finally, these sorted segments are merged to form the final sequence. An illustrative example of parallel merge sort is presented below, where each thread is assigned a number from the unsorted array [34].

Figure 3.5 depicts a CUDA implementation of merge sort using two blocks and four threads per block, showcasing our method's operation. This approach harnesses the parallel processing capabilities of GPUs to expedite the sorting process, thereby enhancing our object detection method's overall performance.

In this example, the *sortBlocks* function first sorts the blocks. This function compares each block element with the adjacent one and sorts them accordingly, forming a sorted group of four elements. This step repeats until the block consists of sorted elements. Following this, the *mergeBlocks* function merges the sorted blocks. It combines the blocks into a larger one, yielding a sorted array. This function continues to be called, doubling the block size each iteration until a single sorted block remains.

3.4 Result & Analysis

We will showcase the benchmark results of our implementation. It's worth noting that our framework tests did not include the training process, only

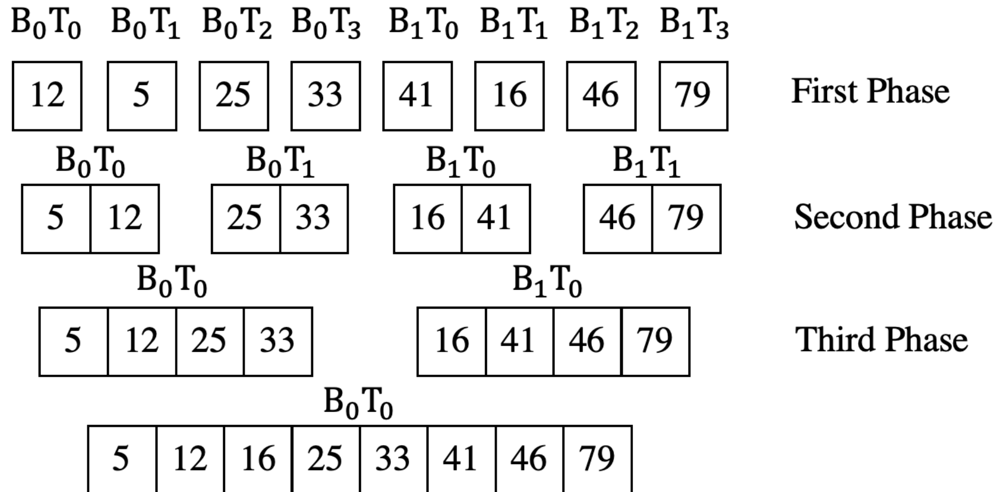

Figure 3.5: Illustration of parallel merge sort with CUDA.

the prediction phase. We directly used the weights trained from the PyTorch version, ensuring a fair comparison between the two implementations. First, we will provide the results for SSD512, followed by a discussion on the performance of the NMS algorithm, where we have paid lots of effort. A summary of our test environment can be found in Table 3.1:

CPU	Intel(R) Xeon(R) CPU E5-2678 v3
GPU	GeForce GTX 1080 Ti
OS	Ubuntu 18.04
CUDA	11.0
cuDNN	8.0

Table 3.1: Test Environment

In this study, we compare our implementation with [73] using the PASCAL VOC2007 dataset [32]. This classic dataset contains 20 categories of objects for identification purposes. We will conduct various analyses on accuracy and speed later on. Our tests were conducted using a GeForce GTX 1080 Ti. This choice was deliberate to ensure the applicability and scalability of our methodology across a wide variety of real-world scenarios. Given that the 1080 Ti is relatively older, we reasoned that if our techniques could

demonstrate effective performance enhancement on this GPU, they will also be likely applicable to newer, more powerful GPUs. Consequently, this guarantees that our approach is future-proof and adaptable, ensuring its relevance and usability as technology progresses.

Another reason for selecting the GeForce GTX 1080 Ti is its larger memory capacity, which facilitates the execution of all computations on the GPU without necessitating data exchanges with the CPU. Initially, we used the GeForce GTX 1050 Ti, but its limited memory size did not allow all computations to run on the GPU exclusively. This requirement for a larger memory capacity is critical to our objective of minimizing data transfers between the CPU and GPU, thus ensuring optimal performance. In addition, it is worth noting that all our implementations utilize single-precision floating-point format (FP32) for computations. This choice is motivated by the balance between computational precision and resource utilization, as FP32 offers sufficient precision for most computer vision tasks while consuming less memory and computational resources compared to higher precision formats.

3.4.1 Accuracy

To verify the effectiveness of our reconstructed SSD512 network, we compare its accuracy with the original work [78]. Precision and recall are two metrics used to evaluate accuracy. The accuracy of each output bounding box can be categorized into four classifications: true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN). Precision is defined as:

$$Precision = \frac{TP}{TP + FP}$$

Here, TP and FP represent the number of true positives and false positives, respectively. The recall is defined as:

$$Recall = \frac{TP}{TP + FN}$$

where FN represents the number of false negatives. Precision indicates how many default boxes correctly represent the target objects' positions among the outputs, i.e., the correctness of the output bounding boxes. Recall refers to the number of target objects captured in the final output. Since items in datasets are divided into multiple classes, another metric called "mean average precision" (mAP) is introduced to represent the overall pre-

cision:

$$mAP = \frac{1}{|classes|} \sum_{c \in classes} Precision(c)$$

where $|classes|$ denotes the number of classes. We conducted an experiment to assess the mAP of SSD512. The original SSD512's performance is provided by the author of the study. The experiment uses the same PASCAL VOC2007 datasets [32] as the SSD publication[78]. The results are compared with the reported value in Table 3.2. The term "proposals" refers to the number of default boxes generated in the final output of SSD512.

Model	Proposals	mAP(%)
Original SSD512	200	75.6
	400	76.4
	1000	76.8
	4000	76.4
Our accelerated SSD512	200	75.2
	400	76.1
	1000	76.4
	4000	76.2

Table 3.2: The mAP comparison of our SSD512 with the original

Experiments were conducted using the PASCAL VOC2007 datasets. "Proposals" refers to the number of default boxes generated in the final output of SSD512. As shown in the right column, our accelerated SSD512 achieves nearly the same level of accuracy as the original one across different proposal amounts. As depicted in Table 3.2, the mAP of our implementation aligns closely with that of the original SSD512, with any differences lying within acceptable margins. Furthermore, the table illustrates that as the number of bounding box proposals increases, the mAP experiences gradual improvement. However, it's noteworthy that an increase in proposal count beyond 4000 sees a bottleneck in mAP improvement, eventually leading to a slight decline. This behavior could be attributed to increased false positives when too many bounding boxes are proposed, adversely affecting precision without improving recall. The comparison of mAP (Mean Average Precision) performances between our CUDA-accelerated SSD512 model and the original PyTorch SSD512 model is detailed in Table 3.2. The varying number of proposals utilized is indicative of the trade-off between computational resources and detection precision.

Upon closer analysis, the disparity in prediction accuracies between the two models may be attributed to the following:

- **Precision Issues**: One of the significant factors that could have influenced the mAP performance of our CUDA-accelerated SSD512 model is the precision of floating-point operations. In our implementation, we used a single-precision floating-point format (FP32), which, while efficient in terms of memory usage and computational speed, can introduce small rounding errors during arithmetic operations. For instance, consider a scenario in our neural network where we are performing a series of multiplications and additions in a layer. This discrepancy seems negligible for this single operation. However, imagine this operation being part of a larger computation involving millions of such operations. The small errors from each operation can accumulate, leading to a more significant deviation in the final result. This accumulated error can manifest in various ways in a deep learning model. For instance, during the forward pass, the computed activations of neurons might slightly deviate from their expected values. During the backward pass, the gradient values used to update the model's weights might be slightly off. These discrepancies, while minor in isolation, can compound over numerous iterations of training, leading to a model that performs slightly differently than expected.

- **Usage of OpenCV Library Functions**: Our CUDA implementation leverages OpenCV for image preprocessing. It's worth noting that certain OpenCV functions, especially those running on the GPU, might have minor numerical discrepancies compared to their CPU counterparts or other libraries. These discrepancies, while typically minuscule, might affect the final prediction results, especially when these functions are used extensively.

- **Randomness**: Despite the lack of training steps in our implementation, the forward propagation process still involves extensive parallel computations. The inherent concurrency in GPU programming can sometimes lead to minor inconsistencies, as the execution order and speed of threads might affect the outcomes. This is especially common when handling operations that require multiple computational stages, such as reduction operations. These inconsistencies can, over time, cause accumulated errors, which might contribute to the lower accuracy observed. Additionally, certain implementations of algorithms for inference in CUDA might involve a degree of non-determinism due to the inherent parallelism of GPU architectures. Consequently, even with

the same input and weights, these operations could lead to slight variations in output between different runs, thus causing slight discrepancies in the final predictions when compared to a more deterministic CPU-based implementation like PyTorch.

- **Hardware discrepancies**: Despite using the same trained weights from the PyTorch model, the hardware discrepancy issue can still be relevant. The computation of neural networks relies heavily on floating-point arithmetic. However, the way floating-point numbers are handled can differ subtly but significantly between CPUs and GPUs, and even among different GPU architectures.

Although the CUDA-accelerated SSD512 model presents slight drops in mAP compared to the original model, the differences lie within reasonable boundaries. In fact, the accelerated model offers near-identical performances while substantially reducing the computational time, demonstrating the feasibility and efficiency of CUDA implementations for deep learning-based object detection models.

3.4.2 Speed

In the ensuing segment of this study, we will unveil the findings derived from our computational speed evaluations, paying particular attention to the performance metrics pertaining to the post-processing stage.

Execution time of each component

SSD512 can be conceptually understood as being partitioned into four key components: pre-processing, feature extraction, proposal layer, and post-processing. The relative time contributions of each of these components within the overall framework have not been widely discussed in the literature, prompting us to perform a detailed measurement in our own experimental setup. The results, as displayed in Figure 3.6, reveal that feature extraction (VGG16) is the most time-consuming process, occupying about 55.38% of the total time. The second most resource-demanding step is post-processing, an area in which we have particularly focused our optimization efforts.

As illustrated in Figure 3.6, a detailed breakdown of the time allocation across the various components of the SSD512 model is provided. It is noteworthy that the process of feature extraction, which currently runs exclusively on the GPU, occupies the majority of the computational time. The remaining components, commonly termed 'common basic processes', are handled by the CPU. These common basic processes, despite being less

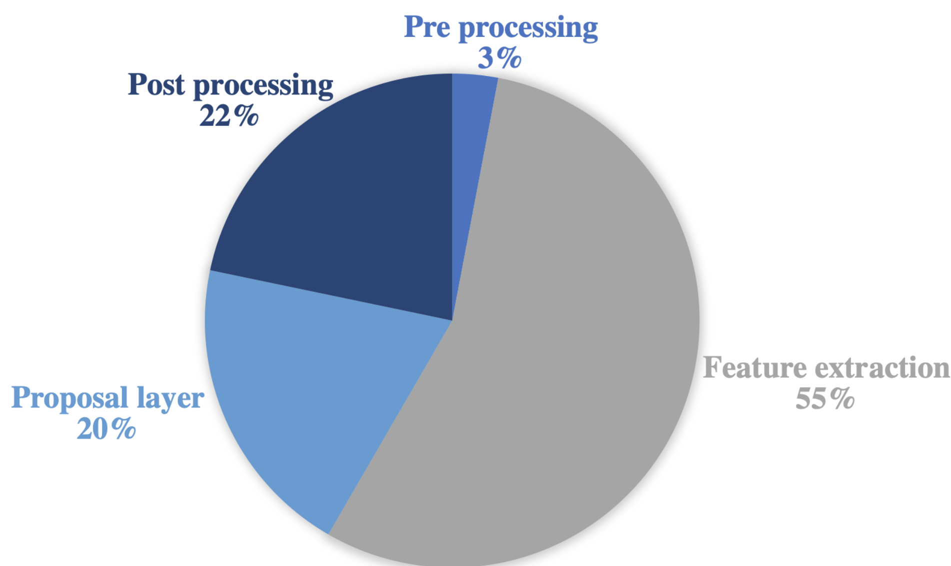

Figure 3.6: Pytorch version of SSD512 time ratio

time-intensive individually, cumulatively represent a substantial portion of the overall processing time. Their computational characteristics and current CPU-bound implementation constitute the primary focus of our research endeavors.

Given the evident time consumption of the feature extraction process and the potential computational power of modern GPUs, migrating these common basic processes from CPU to GPU could indeed be a promising strategy. However, this approach should not be considered a panacea. The feasibility and efficiency of such a transition are contingent on various factors, including the specific computational requirements of each process, the parallelization potential, and the hardware resources available. It is therefore proposed that a systematic, in-depth investigation be carried out to evaluate the potential benefits and drawbacks of porting these common basic processes to the GPU. This would ensure an empirically grounded and technologically sound basis for any subsequent optimizations and could potentially contribute to better performance improvements in the SSD512 model.

Speed Comparison of each component

Figure 3.7 is a comparison of our implementation results with the original SSD512. All parts of our implementation are faster than the original SSD512.

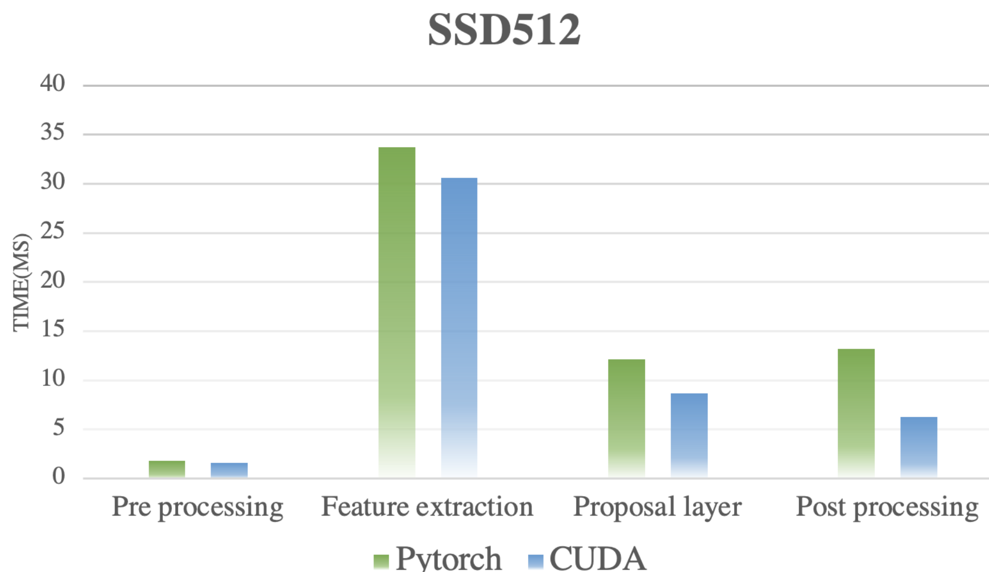

Figure 3.7: Execution time Comparison Result

As illustrated in Figure 3.7, substantial improvements can be observed in the proposal layer and the post-processing layer. By rewriting the feature extraction layer, we also achieved an approximately 9.19% acceleration compared to the original implementation. The proposal layer and the post-processing layer exhibit relatively significant enhancements, with the proposal layer being 28.44% faster and the post-processing layer being 52.61% faster than their original counterparts. Such relatively better speed improvements can be attributed to our refined algorithm and the comprehensive utilization of the GPU.

Although the speed-up in the feature extraction layer is not as pronounced, it still plays a vital role in accelerating the overall SSD512 framework. In summary, our implementation is 22.53% faster than the original version, demonstrating the feasibility of using GPUs to accelerate existing object detection frameworks. This highlights the effectiveness of our approach in enhancing the performance of the SSD512 framework for object detection tasks.

Speed Comparison of NMS algorithm

Initially, our attention will be concentrated on the analysis of the Non-Maximum Suppression (NMS) algorithm, incorporating sorting in our evalu-

ation. This methodical approach aligns with the comprehensive and rigorous standards typically expected in doctoral research.

Proposal/Boxes	CPU(ms)	GPU(ms)	Speed-Up
200/24564	5.82	3.47	×1.68
400/24564	6.37	3.58	×1.78
1000/24564	7.46	4.01	×1.86
4000/24564	13.13	6.47	×2.03

Table 3.3: Execution time Of NMS and sorting(ms)

In table 3.3, We set the threshold to 0.3. The left column shows the number of boxes for the proposal and the number of boxes left after sorting. The middle column shows the results of the tests on the CPU, and the right column shows the results of the GPU tests. In our experiments, we initially had 24,564 bounding boxes, which is also the same number as SSD512. We eliminate low-scoring boxes during the sorting process and subsequently sort the remaining ones. We then select the top 200, 400, 1,000, and 4,000 boxes for evaluation. Since the number of boxes varies, the test results differ accordingly. In our previous comprehensive assessment, we employed data from 4,000 default boxes. The case with 4,000 boxes is the most computationally demanding.

It's noteworthy to mention that as the number of proposals (or bounding boxes pre-sorting) increases, the speed-up factor also rises. This observation can be attributed to the ability of GPUs to exploit parallelism more effectively, thus yielding higher relative performance as the number of simultaneous tasks increases. In a scenario where there are only 200 proposals, the GPU is already 1.68 times faster than the CPU. When the number of proposals increases fourfold to 800, the speed-up factor escalates to 1.78. This trend continues up to 4,000 proposals, where the GPU is over twice as fast as the CPU. One of the intriguing aspects of these results is the demonstration of the CUDA-optimized algorithm's ability to maintain high performance even when the number of proposals is large. This is particularly crucial in real-world scenarios where the number of potential bounding boxes could be quite high, reinforcing the advantage of GPU-based methods in such computational contexts. The robustness of the CUDA-based approach to the number of proposals underlines its superiority and its potential for wider adoption in similar computational challenges.

In the development of our methodology, we strategically leverage one-dimensional blocks in our GPU-accelerated computation. This approach emerged as the most effective solution through a series of rigorous testing sce-

Proposal	200	400	1000	4000	20000
Faster Python[85]	0.67	1.32	2.17	8.06	78.61
Map-reduce[92]	0.28	0.54	1.29	5.69	42.67
Our Method	0.08	0.13	0.66	3.28	28.75

Table 3.4: Execution time of Different NMS(ms)

narios, ensuring the alignment of our computational problem characteristics with the selected model to yield optimal results. The use of one-dimensional blocks enhances the manageability of threads, enabling the efficient utilization of the GPU's computational resources. This approach simplifies the mapping between the data elements and the GPU threads, reducing computational overhead and in turn, boosting overall performance.

Table 3.4 provides a comparative analysis of our method against two other techniques across various proposal volumes. Our proposed method consistently outperforms the other two techniques, providing a convincing demonstration of the effectiveness of our GPU-accelerated approach. Specifically, the results validate our proposition that a GPU-based algorithm, when handling computationally intensive tasks like Non-Maximum Suppression (NMS), can deliver a substantial performance improvement. As the number of proposals escalates, all methods naturally exhibit increased execution time. However, our method's execution time grows at a relatively slower pace, indicating superior scalability, a quality crucial in real-world applications where the number of proposals can be substantial. Moreover, the one-dimensional block structure, despite its simplicity, illustrates an excellent balance of scalability and efficiency, particularly with the escalating number of proposals. However, with the increase in proposal volume, the speed-up ratio tends to taper, largely due to the finite number of available GPU threads. Once the tasks surpass the GPU's thread capacity, they must be queued, causing waiting times that can impact performance.

Despite this minor setback, the overall performance of our method remains superior. Furthermore, this observation highlights potential avenues for further optimization and improvement. Future work could consider the implementation of advanced strategies, such as dynamic resource allocation or more sophisticated scheduling algorithms, to circumvent this limitation and fully exploit the GPU's computational power.

NMS algorithms employing various batch sizes effectively remove related boxes for distinct types of boxes, leading to a relatively significant increase in speed. The different batch size method is incorporated into the NMS algorithm, allowing for the separation of diverse kinds of boxes. This technique is

realized by adding an offset to boxes of different categories. However, as the number of categories increases, more memory is consumed, which in turn, results in a corresponding decrease in speed, as shown in Table 3.5. This trade-off highlights the need for careful consideration when selecting batch sizes in order to optimize both memory usage and processing speed.

Batch Size	CPU(ms)	GPU(ms)	Speed-Up
1	1.32	0.129	×10.23
2	3.16	0.147	×21.50
4	7.51	0.162	×46.36
8	16.68	0.193	×86.42
16	35.93	0.285	×126.07

Table 3.5: Performances of NMS with different batch size(ms)

Batch size, in the context of our implementation, refers to the number of images that are processed simultaneously during the Non-Maximum Suppression (NMS) stage. NMS is a crucial step in object detection algorithms, including our SSD512 model, where it helps in reducing the number of overlapping bounding boxes by keeping only the most probable ones. By increasing the batch size in the NMS algorithm, we process more images at once, potentially speeding up the prediction process. However, it also requires more memory resources. Therefore, choosing an appropriate batch size in the NMS stage is a balance between computational efficiency and resource utilization. Table 3.5 provides an overview of the CPU and GPU performances of NMS across different batch sizes, highlighting the superior efficiency of our proposed method. With a batch size of 1 and 400 proposals, our method gets a speed-up ratio of at least 10. This performance enhancement becomes increasingly evident as the batch size incrementally increases, signifying the effectiveness of our approach in boosting processing speed while maintaining the requisite level of accuracy in object detection tasks. A deeper examination of the data reveals that the execution time of the GPU method does not scale linearly with the batch size. In instances where the batch size is less than 16, the execution time remains relatively stable. This can be attributed to the overhead of kernel launch dominating the processing time when dealing with smaller batch sizes. Simultaneously, as the batch size expands, thread-level parallelism intensifies, launching more threads and facilitating a higher volume of calculations within the same timeframe. This situation contributes to the improved speed of our method, reflecting the powerful capability of GPU computation.

However, when the batch size equals or exceeds 16, the number of threads

launched may surpass the number of threads a GPU can execute concurrently. This saturation point results in an extended execution time due to the need for waiting until more threads are available. Importantly, we adopted a two-dimensional block structure in this scenario, introducing an additional dimension for batches. This strategic adaptation is found to be exceptionally effective, providing an impressive performance boost while dealing with larger batch sizes. The two-dimensional block configuration permits more fine-grained control over thread allocation, thus maximizing parallel processing and enhancing the efficiency of the overall computation. In essence, these results underscore the potential of GPU-accelerated computations in handling large-scale tasks, particularly when coupled with the right configurations, such as our two-dimensional block structure. However, it also hints at the necessity to further explore the trade-offs between the batch size, the overhead of thread launching, and the maximum thread capacity of the GPU, to refine the method and achieve even higher performance.

In the case of CPU-based NMS, each image takes 1.32ms to process when the batch size is 1 and the number of proposals is 400. According to our results in Table 3.3, our approach takes 47.17 ms to perform an image inference with a batch size of 1 and 4000 proposals. If NMS runs on the GPU, it will account for approximately 13.72% of the total processing time. Utilizing the GPU, the NMS can be completed in roughly 3ms, which is a negligible amount of time within the overall pipeline. This represents an improvement in performance speed.

Other Framework

In figure 3.8, we present a comparison of our accelerated SSD512's speed against multiple other target detection frameworks. While creating an apples-to-apples comparison is inherently difficult, we have strived to contextualize our implementation against other prevalent frameworks. Choosing the most effective model isn't a clear-cut process, as real-world applications often necessitate a careful equilibrium between speed and accuracy.

Apart from detector types, one must also contemplate additional elements that can significantly impact performance. These factors include feature extractors (such as VGG16, ResNet and MobileNet), Non-maximum suppression (NMS) IoU threshold, bounding box encoding, and other related considerations. In Figure 3.8, we depict the speed test outcomes for a diverse array of object detection frameworks. The observations reveal that the overall speed of two-stage frameworks[26, 105] tends to lag behind one-stage frameworks[104]. However, these two-stage frameworks typically achieve higher accuracy. Despite the landscape, our implementation showcases a

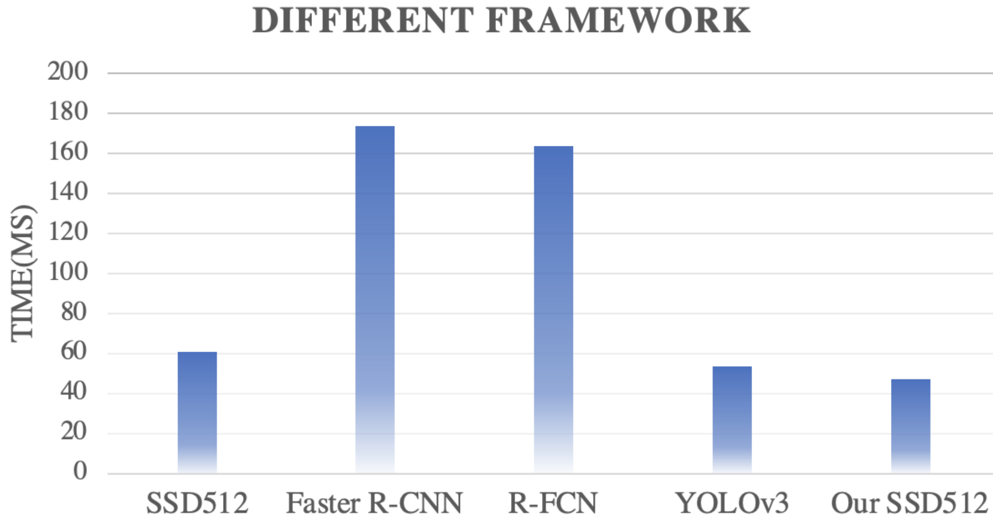

Figure 3.8: Execution time Comparison Result with Other Framework

noteworthy edge in terms of detection speed, earning the distinction of being the fastest among the analyzed frameworks. Please note that this does not declare our approach as universally superior, but merely highlights the effectiveness of our implementation in specific contexts where detection speed is of paramount importance. Furthermore, our approach is not confined to a specific context, and instead, exhibits a high degree of versatility. It can be seamlessly applied to a wide array of detection frameworks, extending its applicability beyond merely object detection.

3.5 Conclusions

Our research, which focused on the GPU acceleration of the SSD512 object detection algorithm, has yielded tangible outcomes. By optimizing common processes found in a variety of detection networks, we have not only enhanced the performance of SSD512 but also outlined a methodology that can be extrapolated to other better detection networks. These include prominent architectures such as Mask R-CNN[46], YOLO[102], and M2det[139], among others.

Our speed-up approach has demonstrated the capability to handle the intricate complexity associated with these detection networks and made an effective trade-off between accuracy and speed. An important insight derived from our analysis is that the complexity of the detection architecture directly

influences the proportion of common processes involved. As such, our approach gains in relevance and value with the escalating complexity of these architectures, paving the way for further enhancements in state-of-the-art object detection systems.

Despite observing improvements in execution time, it is also noteworthy that convolution and fully connected layers still constitute a substantial proportion of the overall processing time. It suggests a fertile ground for further optimizations and improvements. We hypothesize that applying our techniques to a network with less convolution computation could potentially yield a more marked speedup, a prospect that we intend to explore in our future work.

Our tests were predominantly conducted on a relatively powerful GPU, which delivered robust results. However, for broader applicability and commercial viability, it is important to consider GPUs with lower power consumption. While these GPUs may not deliver the same level of performance as high-end GPUs, they offer a better balance between power efficiency and computational power, making them more suitable for a wide array of real-world applications. These applications span across diverse sectors including but not limited to autonomous vehicles, unmanned aerial vehicles, and even edge computing devices. This points to a potentially promising research direction - exploring more power-efficient GPUs and devising tailored strategies to maximize their performance in object detection tasks, which will constitute a significant part of our future endeavors.

In conclusion, the research presented here establishes a solid foundation for GPU-accelerated object detection. It has elucidated the potential for further advancements and set a clear direction for future explorations in this domain. As technology advances and object detection algorithms become more complex, our approach provides an essential blueprint for harnessing the power of GPUs to achieve unprecedented levels of performance.

Chapter 4

Optimization for the Swin Transformer

4.1 Introduction

The field of computer vision has witnessed remarkable advancements in recent years, fueled by the rapid progress of deep learning techniques. Convolutional neural networks (CNNs) have served as the foundation of computer vision, achieving significant success in various tasks such as image classification [66, 115, 48], object detection [42, 105, 102, 69, 142], semantic segmentation [82], and video recognition [59, 127]. Nevertheless, the recent advent of transformers [124], initially designed for natural language processing tasks, has introduced new opportunities in the computer vision domain, challenging the supremacy of CNNs. Following the successful application of the AlexNet [66] in image classification tasks, the field of convolutional neural networks (CNNs) experienced an explosion of innovative research. This period saw the development of several prominent backbone networks such as GoogLeNet [120], VGG [115], and ResNet [48]. Concurrently, the domain of object detection frameworks also witnessed significant advancements. Techniques such as Faster R-CNN[105] and Mask R-CNN[46] emerged as two-stage detectors, offering a more accurate albeit slower alternative to single-stage detectors like YOLO [102] and SSD [78].

Despite the success of CNNs in the field of computer vision, recent research has shown that Vision Transformers (ViT) [31]have the potential to surpass CNNs' performance in various visual applications [3, 17, 7, 117, 80]. Introduced by Vaswani et al. [124], Transformers have revolutionized the field of natural language processing (NLP) and have recently shown promise in computer vision tasks.

Initially developed for machine translation, the Transformer's ability to model long-range dependencies has proven beneficial for a variety of tasks. Further advancements led to the development of Transformer-based pre-trained models (PTMs) [96], which have demonstrated state-of-the-art performance in numerous NLP tasks.

The successful application of Transformers in NLP tasks prompted researchers to explore their potential in other domains. Dosovitskiy et al. [31] pioneered this initiative by introducing the Vision Transformer (ViT), which demonstrated competitive performance on image classification tasks. Subsequent research expanded the application of Transformers to other computer vision tasks, resulting in architectures such as DETR [17] for object detection, DeiT [122] for image classification, and TimeSformer [8] for video recognition. Despite their remarkable results, these architectures often face scalability issues and high computational complexity. Addressing these limitations, Liu et al. [80] introduced the Swin Transformer, a hierarchical vision transformer that employs a local window-based self-attention mechanism and a shifted window strategy. This innovative approach has led to state-of-the-art performance across a range of computer vision tasks and benchmarks, exhibiting strong scalability and adaptability. As such, the Swin Transformer represents a promising avenue for future research and applications in the field of computer vision.

In the preceding chapter, we discussed the employment of GPUs to accelerate object detection frameworks, primarily those employing backbone networks reminiscent of CNNs. However, with the emerging dominance of the Transformer model, which has demonstrated considerable prowess in object detection, our focus in this chapter pivots towards the Transformer model, specifically the Swin Transformer. Our objective initially involved optimizing the model to a degree, all the while preserving its inherent strengths, and subsequently utilizing the GPU to alleviate the computational speed challenges. The Swin Transformer was selected for this purpose, given its high-performance characteristics and impressive efficacy in a variety of computer vision tasks. The presence of parallel computation possibilities within its structure allows for acceleration, thereby rendering it an ideal choice for our studies. This thoughtful strategy led to effective optimization without compromising the core benefits of the Swin Transformer model.

In this chapter, we build upon the Pyramid Swin Transformer, which employs different-size windows in the Swin Transformer architecture to enhance its performance in image classification and other computer vision tasks, achieving an effective trade-off between accuracy and computation complexity. The Pyramid Swin Transformer addresses the problem of the lack of connections among windows on a large scale in the original Swin Trans-

former. To enhance training efficiency and detection speeds, we have honed the acceleration of the underlying window-based Multi-head Self Attention (W-MSA) mechanism which is the most important basement part within the Swin Transformer and Pyramid Swin Transformer models. Given the innate potential of W-MSA for parallel computation, we identified GPU utilization as an effective strategy for its acceleration optimization, looking forward to striking an optimal balance between speed and accuracy.

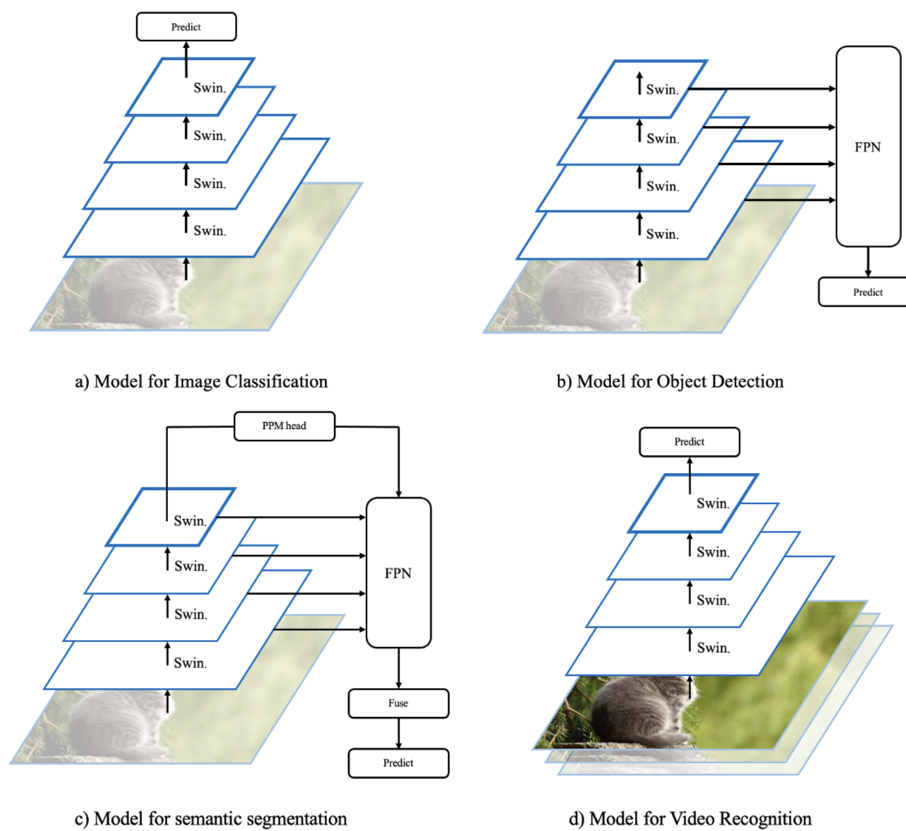

Figure 4.1: Pyramid Swin Transformer

In figure 4.1, a) model is used for image classification, b) model is used for object detection, c) model is used for semantic segmentation, d) model is used for video recognition.

Our goal is to test our Pyramid Swin Transformer on a wider range of computer vision tasks, demonstrating its potential for various vision applications and its superior performance, by implementing multiple windows of varying sizes on an extensive feature map, we construct a layered hierarchy of windows. In this arrangement, a single window in the upper layer encapsulates

the features of four windows in the immediate lower layer. This progression strengthens the interconnections among the windows, thereby enhancing the overall representational capability of the model. Furthermore, the fundamental aim of this work is to enhance detection results while maintaining optimal speed. As our Pyramid Swin Transformer architecture adds to the computational burden, it becomes crucial to assure the efficiency and competitiveness of our model for training and detection tasks. With this in mind, we have utilized the inherent parallelism of the Pyramid Swin Transformer's W-MSA to formulate a CUDA-enabled W-MSA algorithm. By leveraging the CUDA-based W-MSA algorithm that exploits the parallelism of the core W-MSA operation within the Pyramid Swin Transformer, we are ensuring the competitiveness of our model, leading to more effective and efficient training and detection procedures. Our experiments demonstrate the improved performance achieved by the new architecture across four vision tasks: image classification on ImageNet [108], object detection on COCO[77], semantic segmentation on ADE20K[141], and video recognition on Kinetics-400[60], as shown in Figure 4.1. Moreover, our GPU-accelerated variant of W-MSA outpaces the performance of the original W-MSA implementation in terms of speed.

4.2 Related Work

In this section, we provide an overview of the key advancements and architectures that have contributed to the development of our Pyramid Swin Transformer. We begin by discussing Feature Pyramid Networks, which have been instrumental in building hierarchical feature representations. Next, we discuss the emergence of Vision Transformers, which have shown great promise in various computer vision tasks. Finally, we delve into the Swin Transformer, a hierarchical vision transformer that has achieved state-of-the-art performance in several vision benchmarks.

4.2.1 Feature Pyramid Network

Feature Pyramid Networks (FPNs) [75] are a popular and versatile neural network architecture that addresses the challenge of scale variation in images. They have been successfully applied to a wide range of computer vision tasks, such as object detection, semantic segmentation, and instance segmentation. In this subsection, we provide an in-depth overview of FPNs, discussing their motivation, design principles, and applications in various computer vision tasks.

Scale variation is a fundamental challenge in computer vision, as objects in real-world images can appear at vastly different sizes and resolutions. Traditional convolutional neural networks (CNNs) often struggle with this issue, as they are designed to capture local patterns and may not be robust to changes in scale. To address this problem, researchers have proposed various approaches, such as image pyramids [1], scale-invariant feature transform (SIFT) [83], and more recently, deep learning-based methods like multi-scale feature learning [113] and spatial pyramid pooling (SPP) [47]. However, these methods either require a large amount of computation or fail to fully exploit the rich hierarchical feature representations learned by deep CNNs.

FPNs were proposed as a solution to these shortcomings, building on the strengths of deep CNNs while effectively handling the issue of scale variation. By combining low-level features with high-level semantic information, FPNs can efficiently process objects of different scales in an image.

Design Principles

FPNs consist of a bottom-up pathway, a top-down pathway, and lateral connections. The bottom-up pathway is essentially a deep CNN that computes a feature hierarchy with increasing levels of abstraction. At each level, the spatial resolution is downsampled by a factor of 2, and the number of channels is typically doubled to maintain a constant time complexity. This results in a pyramid of feature maps, with each level corresponding to a different scale.

The top-down pathway aims to refine and propagate high-level semantic information to lower levels of the pyramid. This is achieved by upsampling the feature maps at each level by a factor of 2 and then combining them with the corresponding feature maps from the bottom-up pathway via lateral connections. The lateral connections are implemented using 1×1 convolutional layers, which serve to reduce the number of channels in the bottom-up feature maps to match the top-down feature maps. The combination of the upsampled top-down feature maps and the lateral connections is performed using element-wise addition, effectively fusing the high-level semantic information with the fine-grained spatial details.

The resulting feature pyramid can be used as a multi-scale feature representation for various computer vision tasks. For instance, in object detection, the pyramid can be fed into a region proposal network (RPN) [105] and a detection head to predict bounding boxes and class labels for objects at different scales. Similarly, in semantic segmentation, the pyramid can be used to generate dense pixel-wise predictions by upsampling and combining the feature maps at different levels.

Applications

FPNs have been widely adopted in various computer vision tasks due to their effectiveness in handling scale variation:

Object Detection: FPNs were initially introduced for object detection in the context of the Faster R-CNN framework [75]. By incorporating FPNs, the detection performance significantly improved, especially for small objects. FPNs have since become a standard component in many state-of-the-art object detection architectures, such as RetinaNet [76], Cascade R-CNN []cai2018cascade, and Mask R-CNN [46].

- **Semantic Segmentation:** FPNs have been successfully applied to semantic segmentation tasks as well. In the context of DeepLab [18], FPNs were used to generate multi-scale feature representations that improved the model's ability to handle objects of varying scales. Similarly, PSPNet [138] employed a pyramid pooling module that built upon the FPN architecture to capture global context information for semantic segmentation.

- **Instance Segmentation:** Instance segmentation is another area where FPNs have demonstrated remarkable performance. Mask R-CNN [46], a widely-used instance segmentation framework, integrated FPNs into its architecture to achieve state-of-the-art results on the COCO dataset [77].

- **Panoptic Segmentation:** Panoptic segmentation is a task that combines semantic and instance segmentation, aiming to assign a class label and instance ID to each pixel in an image. FPNs have been utilized in panoptic segmentation architectures like Panoptic FPN [62] and UPSNet [131], where they help provide multi-scale feature representations and improve the overall segmentation performance.

- **Pose Estimation:** FPNs have also been employed in the field of human pose estimation. In the context of multi-person pose estimation, the FPN architecture has been incorporated into models like HigherHRNet [19] to capture multi-scale human key points, improving the estimation accuracy for both small and large-scale human instances.

In summary, Feature Pyramid Networks have demonstrated their effectiveness in handling scale variation and have been widely adopted in various computer vision tasks. Their ability to fuse high-level semantic information with fine-grained spatial details makes them a powerful tool for addressing the challenges posed by objects of different sizes in real-world images.

4.2.2 Vision Transformer

The Vision Transformer (ViT) has gained significant attention in the computer vision community due to its competitive performance in various vision tasks. The underlying principle of ViT is the Transformer model [124], which was originally introduced for natural language processing (NLP) tasks. In recent years, the Transformer model has been successfully adapted to handle computer vision problems, showing promising results and even outperforming traditional convolutional neural networks (CNNs) in some cases.

Transformer Model

The Transformer model, introduced by Vaswani et al. [124], revolutionized the field of natural language processing by addressing the limitations of recurrent neural networks (RNNs) and convolutional neural networks (CNNs) in capturing long-range dependencies in input sequences. Unlike RNNs and CNNs, which process sequences iteratively or through local receptive fields, the Transformer model employs a self-attention mechanism to directly model dependencies between any two elements in an input sequence, regardless of their distance. The core component of the Transformer model is the Transformer block 4.2, which consists of two main sub-layers: a multi-head self-attention mechanism and a position-wise feed-forward network. The multi-head self-attention mechanism enables the model to capture different aspects of the relationships between elements in an input sequence by computing weighted sums of the input elements. The weights for each element are calculated based on the similarity between the element's query, key, and value vectors. This allows the model to dynamically focus on different parts of the input sequence, thereby capturing long-range dependencies and complex interactions between elements. To provide information about the relative positions of elements in the input sequence, the Transformer model incorporates positional encodings into its input embeddings. These encodings are added to the input embeddings before they are processed by the multi-head self-attention mechanism, thus allowing the model to consider the positions of the elements when computing attention weights.

Following the multi-head self-attention mechanism, the output is then passed through a position-wise feed-forward network, which consists of two fully connected layers with a non-linear activation function (e.g., ReLU) in between. This network further processes the information extracted by the self-attention mechanism and helps refine the representations. The Transformer model is built by stacking multiple Transformer blocks, forming a deep architecture capable of learning complex relationships and generating

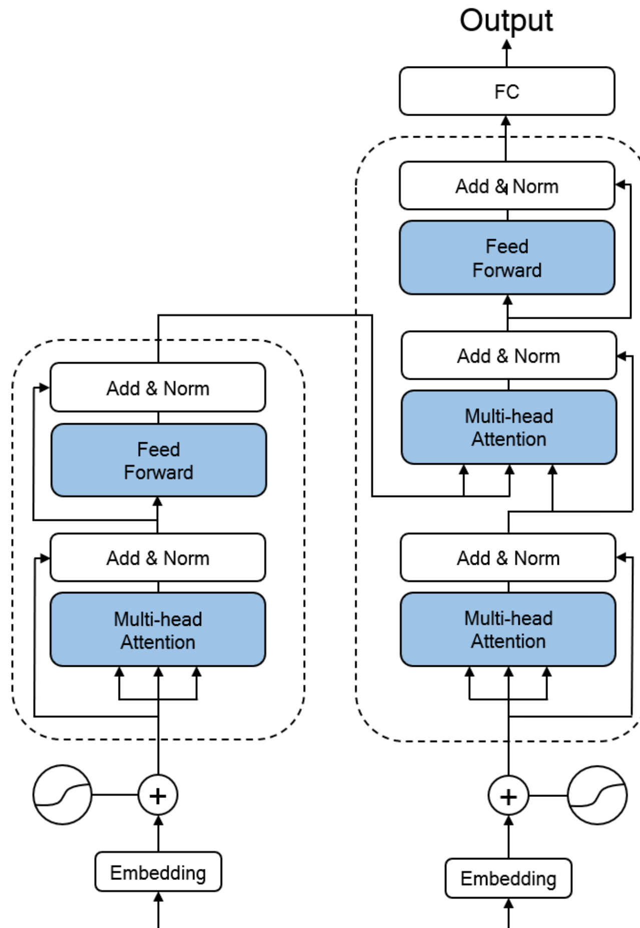

Figure 4.2: Transformer block

rich representations. The model is trained end-to-end using gradient-based optimization techniques. Since its introduction, the Transformer model has been widely adopted and adapted for various tasks, not only in natural language processing but also in computer vision, where it has been used to replace or complement traditional CNN-based architectures.

Self-attention Mechanism The self-attention mechanism, also known as the transformer model in some contexts, is a powerful technique used in many modern machine learning models. It is particularly popular in the field of natural language processing, where it forms the backbone of models like BERT[30], GPT[97], and Transformers[124], but has also been applied with great success in vision tasks.

Self-attention is a method of assigning importance weights to input ele-

ments. It is called "self" attention because the weights are determined based on the input itself, rather than on any external context or learned parameters. This allows the model to focus on the most important or relevant parts of the input when making predictions, rather than treating all input elements equally. In the self-attention mechanism, each element in the input (which could be words in a sentence, pixels in an image, or nodes in a graph) is compared with every other element to compute a score. This score is then used to weigh the contribution of each element to the output. This enables the model to capture interactions between elements that are far apart in the input sequence, overcoming the limitations of models that only consider local context. In essence, the self-attention mechanism provides a flexible and powerful way for models to automatically learn to focus on the most important parts of their input, leading to more accurate and robust predictions. As our understanding of this mechanism continues to grow, it is likely to play an increasingly important role in the development of advanced machine-learning models. The self-attention mechanism is defined as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V, \tag{4.1}$$

where Q, K, and V are the query, key, and value matrices, respectively, and d_k is the dimension of the key vectors. These matrices are derived from the input through linear transformations using learned weight matrices:

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V, \tag{4.2}$$

where W_Q, W_K, and W_V are the learned weight matrices for queries, keys, and values, respectively, and X is the input matrix.

Figure 4.3, the self-attention mechanism computes a weighted sum of the input values, where the weights are determined by the compatibility between the queries and keys. This allows the model to focus on different parts of the input sequence, effectively capturing the dependencies between elements.

Multi-Head Attention The Multi-Head Attention mechanism is an advanced variant of the self-attention mechanism that aims to expand the model's ability to focus on different types of information in the input. In standard self-attention, each input element interacts with every other element to produce an attention score, which is used to weigh the contribution of each element to the output. While this allows the model to capture complex interactions within the input, it also means that the model uses the

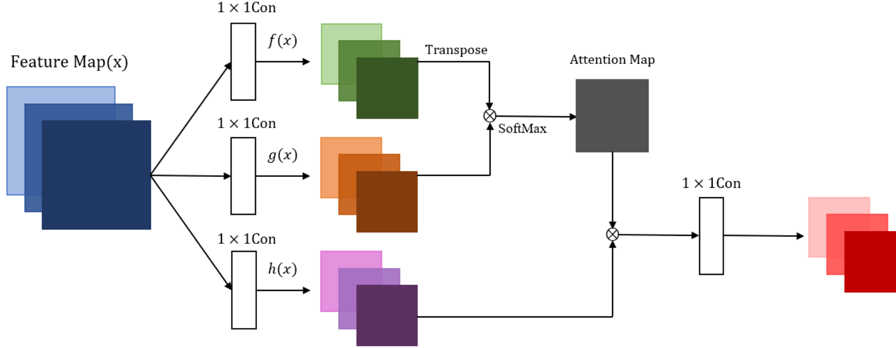

Figure 4.3: Self Attention Architecture

same method to consider all types of interaction. To capture different aspects of the input data, the Transformer model uses multi-head attention, which consists of several self-attention mechanisms running in parallel:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O, \tag{4.3}$$

where $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$, and W_i^Q, W_i^K, W_i^V, and W^O are learned weight matrices.

The multi-head attention mechanism, as shown in figure 4.4, allows the model to learn different types of dependencies and interactions between the elements in the input sequence.

The Multi-Head Attention mechanism addresses this by splitting the attention process into multiple parallel 'heads'. Each head performs its own self-attention computation, effectively allowing the model to learn different types of interaction in parallel. The outputs of each attention head are then concatenated and linearly transformed to produce the final output. This allows the model to consider and combine different types of information in the input, leading to a richer understanding of the input data. The number of heads in the Multi-Head Attention mechanism is a hyperparameter that can be tuned based on the specific task and data. In practice, increasing the number of heads can often lead to improved performance, as it allows the model to consider more types of interaction simultaneously. The Multi-Head Attention mechanism is a key component of many modern machine learning models, especially in the field of natural language processing. It allows these models to capture a more nuanced understanding of their input data, leading to improved performance on a wide range of tasks.

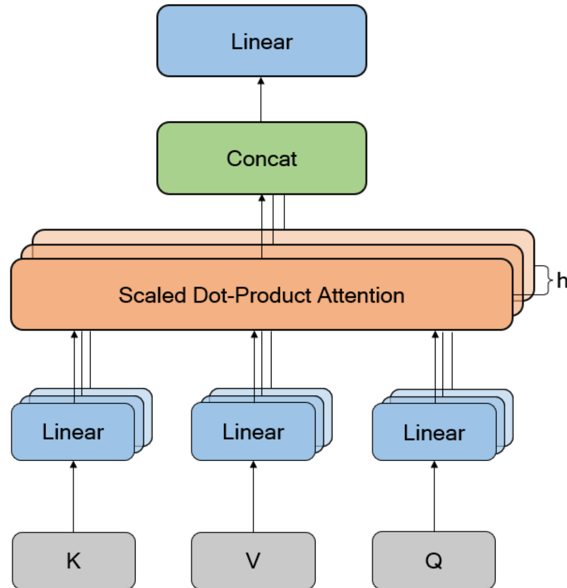

Figure 4.4: Multi-head Attention Architecture

Vision Transformer Model

In the context of NLP, the input is typically a sequence of word embeddings, but in the case of ViT, the input is a sequence of image patch embeddings. To create the input sequence for ViT, an image is divided into non-overlapping patches, and each patch is linearly embedded into a flat vector. The sequence of patch embeddings is then processed by the Transformer model to capture both local and global contextual information.

Dosovitskiy et al. [31] first demonstrated the applicability of the Transformer model to vision tasks by introducing the ViT architecture 4.5. The authors showed that ViT achieves competitive performance in image classification tasks when trained on large-scale datasets. Furthermore, the ViT architecture benefits from transfer learning, where a model pre-trained on a large dataset can be fine-tuned for specific tasks or smaller datasets, resulting in better performance compared to models trained from scratch.

To incorporate positional information, ViT adds learnable positional embeddings to the patch embeddings. This approach allows the model to learn spatial relationships between patches, which is crucial for capturing the structure of the visual data.

The ViT architecture is composed of a series of Transformer blocks, each containing a layer normalization, a multi-head self-attention mechanism, another layer normalization, a position-wise feed-forward network, and a resid-

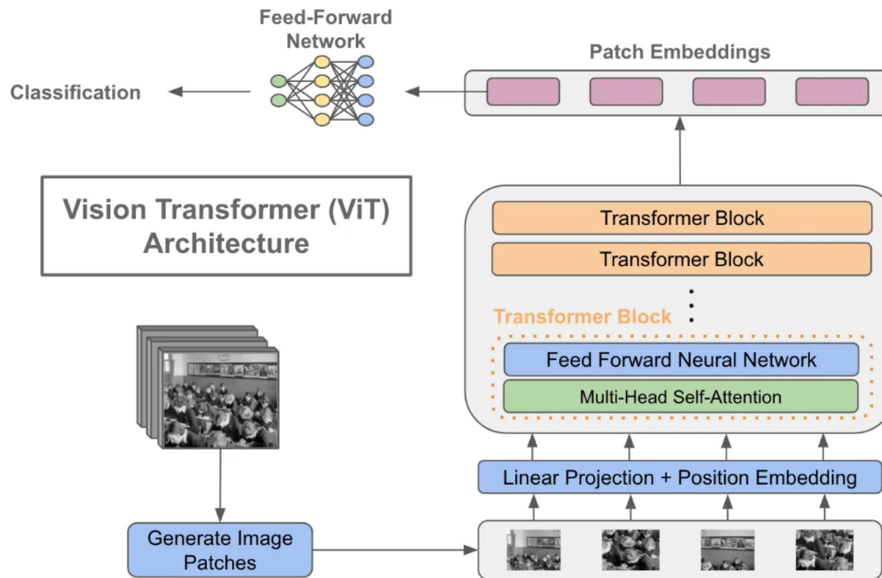

Figure 4.5: Vision Transformer[15]

ual connection. These components are stacked to form a deep transformer network that processes the input sequence of patch embeddings.

Despite its simplicity and effectiveness, the ViT architecture has a few drawbacks. One of the main challenges is the quadratic complexity of the self-attention mechanism with respect to the input sequence length, which can lead to high computational costs and memory requirements. This issue becomes more pronounced as the input resolution or the number of patches increases.

To address this challenge, several variants of the ViT architecture have been proposed, such as the DeiT [122], which employs data-efficient training techniques to achieve better performance with smaller datasets, and the T2T-ViT [134] that introduces a token-to-token self-attention mechanism to reduce the complexity of the attention computation. In summary, the Vision Transformer has emerged as a powerful and flexible model for various computer vision tasks. Its success has inspired a wave of research exploring the use of Transformer models for tasks such as object detection [17], semantic segmentation [140], and video recognition [8]. The development of more efficient vision transformer architectures and the combination with other models or techniques will likely continue to shape the future of computer vision research and applications.

4.2.3 Swin Transformer

The Swin Transformer, proposed by Liu et al. [80], is an innovative approach that addresses the limitations of traditional Vision Transformers, enabling more efficient and scalable models for various computer vision tasks. This section provides a more detailed overview of three key components of the Swin Transformer that contribute to its performance: local window-based self-attention, shifted window strategy, and hierarchical structure.

Window-based Multi-head Self-attention

The window-based self-attention mechanism in the Swin Transformer is designed to address the computational and memory limitations associated with the global self-attention mechanism used in standard Vision Transformers. In this part, we will provide a more detailed explanation of the mechanism and its advantages.

In a traditional Transformer model, the self-attention operation computes the response at a position in a sequence by considering all positions in the entire sequence, resulting in a quadratic computational complexity of $O(n^2)$, where n is the number of input tokens. This complexity becomes a bottleneck when dealing with large input sizes and deeper architectures, which are common in computer vision tasks.

To overcome this challenge, the Swin Transformer introduces a window-based self-attention mechanism. Instead of computing self-attention over the entire input sequence, the mechanism partitions the input feature map into non-overlapping windows and computes multi-head self-attention within each window independently. This approach reduces the computational complexity from $O(n^2)$ to $O(nw^2)$, where n is the number of patches and w is the window size. However, the expression $O(nw^2)$ seems to be a simplification. The detailed computational complexity of window-based multi-head self-attention (W-MSA) is $O(4hwC^2 + 2M^2hwC)$, where h and w are the height and width of the image in terms of patches, C is the number of channels, and M is the window size. This complexity is linear with respect to the number of patches when window size M is fixed, which can be seen as a significant reduction from the quadratic complexity of global self-attention. Furthermore, the window-based self-attention mechanism can be combined with the multi-head self-attention mechanism to enable the model to learn different features at different positions within the window. In a multi-head self-attention setup, the self-attention operation is computed independently for multiple heads, and their outputs are then concatenated and linearly transformed to produce the final output. By leveraging window-based self-

attention, the Swin Transformer effectively addresses the computational and memory limitations of global self-attention, enabling it to scale efficiently to large input sizes and deeper architectures commonly encountered in computer vision tasks.

Shifted Window-based Multi-head Self-Attention

The shifted window-based self-attention figure 4.6 mechanism is another crucial component of the Swin Transformer, designed to address the limited receptive field issue introduced by local window-based self-attention. This section provides a more detailed explanation of the shifted window-based self-attention mechanism and its benefits.

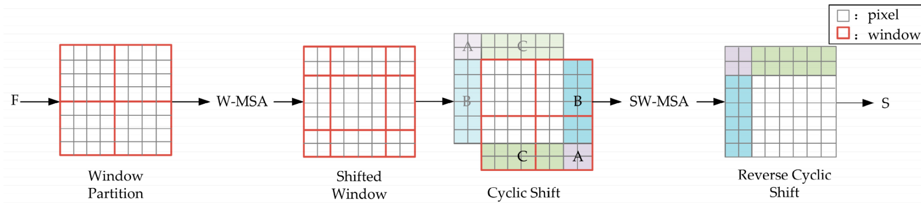

Figure 4.6: Shifted Window-based Self-Attention

As mentioned earlier, local window-based self-attention divides the input feature map into non-overlapping local windows, and self-attention is computed independently within each window. While this approach effectively reduces the computational complexity, it also limits the receptive field of the model, potentially hindering its ability to capture long-range dependencies and context information. To address this limitation, the Swin Transformer incorporates a shifted window-based self-attention mechanism. The key idea behind this mechanism is to shift the windows by half the window size in both the vertical and horizontal directions at alternating layers of the network. This shifting operation ensures that each token's receptive field expands across multiple layers, ultimately covering the entire feature map and allowing the model to capture long-range dependencies.

By integrating the shifted window-based self-attention mechanism, the Swin Transformer effectively addresses the limited receptive field issue introduced by local window-based self-attention, enabling the model to capture both local and long-range contextual information efficiently. This mechanism allows the Swin Transformer to achieve state-of-the-art performance in various computer vision tasks, such as image classification, object detection, semantic segmentation, and video recognition while maintaining computational efficiency and scalability.

Hierarchical Structure

The Swin Transformer's hierarchical structure is designed to efficiently handle objects and features of varying sizes and scales in the input image. This structure is inspired by traditional convolutional neural networks (CNNs) that leverage pooling layers to create a hierarchy of feature maps at different resolutions. In the Swin Transformer, a similar hierarchical approach is implemented using non-overlapping patches and multi-stage processing. At the beginning of the Swin Transformer, the input image is divided into non-overlapping patches at multiple scales, with each scale constituting a different level of the hierarchy. The patches at each scale are transformed into linear embeddings, which serve as the input tokens for the transformer layers.

The Swin Transformer processes these input tokens in a multi-stage manner. Each stage consists of a series of transformer layers that process the tokens at a specific hierarchical level. The tokens are then aggregated and downsampled to form the input tokens for the next stage. The downsampling operation is performed using a patch merging layer, which merges the neighboring tokens in a non-overlapping manner to create new tokens with half the spatial dimensions. This process is repeated across multiple stages, resulting in a hierarchy of token embeddings at progressively lower resolutions. The hierarchical structure of the Swin Transformer enables the model to efficiently handle objects and features at various scales in the input image. This multiscale processing provides an effective mechanism for capturing both local and global context information, ultimately enhancing the model's performance on tasks with diverse scale requirements, such as object detection and semantic segmentation. Moreover, the hierarchical structure allows the Swin Transformer to adapt and scale to different input resolutions and task complexities, making it a versatile and powerful architecture for a wide range of computer vision applications.

4.2.4 Mask R-CNN

Mask R-CNN [46] is an influential model for object detection and instance segmentation, enhancing the Faster R-CNN[105] framework with an additional layer for outputting binary masks for detected objects. This makes Mask R-CNN particularly effective for tasks that not only require identifying and localizing objects but also require a detailed understanding of their shapes.

The Mask R-CNN model comprises two central stages. The first part is a deep convolutional network, often a ResNet[48], acting as the backbone for feature extraction. This network is followed by the Feature Pyramid Net-

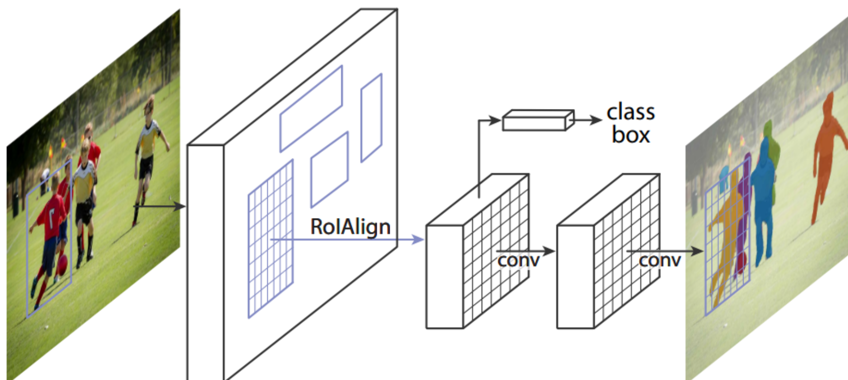

Figure 4.7: Mask R-CNN[46]

work (FPN)[75], a multi-scale architecture that enhances the model's ability to detect objects at varying scales. The second part of Mask R-CNN involves two subnetworks that function atop the feature pyramid - the Region Proposal Network (RPN) for proposing candidate object bounding boxes, and the box heads for classifying and refining these boxes. The RPN performs a comprehensive scan across the feature map output of the previous layer, proposing regions of different scales and aspect ratios as potential object bounding boxes, and scoring each based on its object likelihood. These proposals are then reshaped and funneled into the box head subnetworks.

In the box head, each proposed region undergoes three parallel computations: class prediction, bounding box refinement, and mask prediction. The class prediction uses a fully connected layer to output class scores. The bounding box refinement applies bounding box regression to adjust the coordinates of the proposed region for more accurate object localization. The mask prediction, however, introduces a fully convolutional network to output a binary mask for each class.

Mask R-CNN innovatively addresses the issue of spatial quantization in RoI features. Mask R-CNN introduces RoIAlign, which correctly aligns the extracted features with the input, avoiding any misalignment and enabling the network to learn to predict masks that are spatially coherent with the RoI. Mask R-CNN has achieved wide recognition in object detection and instance segmentation due to its effective design and comprehensive multi-task training. Although simple in implementation, it represents a powerful tool for a range of computer vision applications.

4.2.5 UPerNet

Unified Perceptual Parsing Network or UPerNet [129], is a versatile and robust deep learning framework designed for a variety of pixel-level prediction tasks, including semantic segmentation, object detection, and instance segmentation. This network aims to address these tasks in a unified, holistic manner, minimizing the need for task-specific architectures and creating a more general solution.

UPerNet builds upon the Feature Pyramid Network (FPN)[75] used in models like Mask R-CNN. It enhances the FPN with a top-down pathway and lateral connections to provide multi-scale feature representation, which is crucial for pixel-level prediction tasks that deal with objects of varying sizes and contexts.

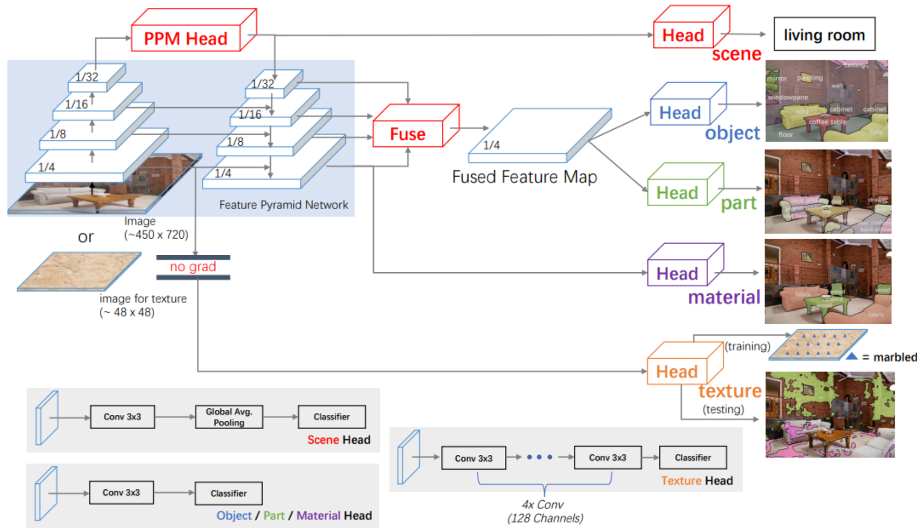

Figure 4.8: UPerNet[129]

The main architecture of UPerNet can be divided into three parts: a backbone network for feature extraction, a Pyramid Pooling Module (PPM)[47] for multi-scale global context representation and a sequential fusion module for feature fusion. UPerNet typically employs a ResNet[48] backbone, although it can be adapted to work with other types of deep convolutional networks. The backbone network is tasked with extracting hierarchical features from the input image. These features, especially from the last few stages of the backbone, carry rich semantic information and are forwarded to the PPM and the sequential fusion module. The PPM in UPerNet is adapted from the Pyramid Scene Parsing Network (PSPNet), a model renowned for

its strong performance in semantic segmentation tasks. The PPM captures global context information by applying pooling operations at different scales and concatenating the resulting features. This operation effectively extracts context cues of various scales from the entire scene, providing a more comprehensive understanding of the global context.

After obtaining multi-scale features from the PPM, UPerNet applies a sequential fusion module for feature fusion. Unlike the FPN, which adopts a top-down pathway and lateral connections for feature fusion, UPerNet performs fusion in a sequential manner. Starting from the deepest stage, UPerNet gradually fuses the features of each stage with the upsampled features from the previous stage. Through this approach, UPerNet preserves rich high-level semantic information while incorporating lower-level details, producing a full-resolution feature map that is beneficial for pixel-level prediction.

In essence, UPerNet provides a unified solution to various perceptual parsing tasks by combining a strong feature extraction backbone, a global context representation module, and a sequential feature fusion scheme. This integration enables UPerNet to handle a wide range of tasks that require both local details and global context understanding, making it a powerful and versatile tool for pixel-level prediction tasks. The architecture of UPerNet reflects the growing trend in computer vision research towards more unified, multi-task frameworks that can leverage shared representations for better performance.

4.2.6 Adam Optimization Algorithm

Adam[61], short for Adaptive Moment Estimation, is an optimization algorithm used extensively in the field of deep learning. It was first proposed by Kingma and Ba in the paper "Adam: A Method for Stochastic Optimization" in 2015.

Adam is a method that computes adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past squared gradients v_t like Adadelta and RMSprop, Adam also keeps an exponentially decaying average of past gradients m_t, similar to momentum:

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$$
$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$$

where: - m_t and v_t are estimates of the first moment (the mean) and the second moment (the uncentered variance) of the gradients respectively, -

β_1 and β_2 are exponential decay rates for these moving averages, - g_t is the gradient at time step t.

As m_t and v_t are initialized as vectors of 0's, the authors of Adam observe that they are biased towards zero, especially during the initial time steps, and especially when the decay rates are small. They counteract these biases by computing bias-corrected first and second moment estimates:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

They then use these to update the parameters, which yields the Adam update rule:

$$\theta_{t+1} = \theta_t - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

where: - θ_{t+1} are the updated parameters, - α is the step size or learning rate, - ϵ is a small number to prevent any division by zero in the implementation (default value is suggested to be 10^{-8}).

Adam has been shown to combine the advantages of two popular stochastic gradient descent advancements: Root Mean Square Propagation (RMSProp) and Adaptive Gradient Algorithm (AdaGrad). This makes it an efficient and effective choice for non-convex optimization problems often encountered in deep learning.

4.3 Pyramid Swin Transformer

Swin Transformer effectively addresses the multi-scale and computational complexity challenges in vision Transformer(ViT)[31], however, it also introduces new issues due to its implementation of window-based multi-head self-attention, which makes the windows independent from each other.

Although the authors have introduced shifted window-based multi-head self-attention to facilitate connections among specific windows, an inadequate information exchange persists between certain windows, especially in large-scale dimensions, as depicted in Figure 4.9. Even when the shift window operation is applied, each window can only connect to four windows post-reintegration. This restricts the window's ability to associate with any of the other five windows, impacting the overall performance of the model. This

example only scratches the surface. At larger scales of feature maps, due to the employment of fixed-size windows, this situation worsens, resulting in more windows losing connectivity with others.

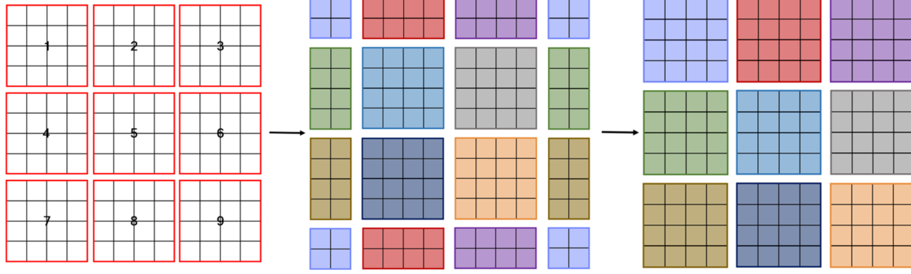

Figure 4.9: Swin Transformer architecture problem

Our model architecture is primarily inspired by and built upon the Swin Transformer [80]. As depicted in Figure 4.10, our design for object detection introduces an additional feature pyramid component and implements crucial modifications to the original Swin Transformer structure. Using Pyramid Swin-R as an instance, we leverage a hierarchical network configuration. In this setup, the first stage possesses the largest dimensions (64×64). At this stage, we fragment the feature map into four distinct window types ($16 \times 16, 8 \times 8, 4 \times 4, 2 \times 2$), each corresponding to a different window size of $4 \times 4, 8 \times 8, 16 \times 16$, and 32×32, respectively.

The window of size 32×32 corresponds to a self-attention span of 1024, which substantially increases the computational complexity. This increase in complexity is due to the nature of multi-head self-attention computation in Transformer-based models. In multi-head self-attention, for each position in the input, the model computes a weighted sum of all positions in the input, with the weights determined by a compatibility function of the input at those two positions. This means the model needs to compute and store values for every pair of input positions, leading to a time and space complexity of $O(n^2)$, where n is the number of positions or the self-attention span. In the case of a 32×32 window, the self-attention span is 1024, meaning the model needs to compute and store 1024 * 1024 = 1,048,576 values for each head in the multi-head self-attention, and this computation is done for every position in the input. As the number of heads or the self-attention span increases, the computational cost grows quadratically, leading to a substantial increase in computational complexity. This is the reason why we limit the use of the 32×32 window size to the first stage in our Pyramid Swin Transformer model to balance the computational demands with the performance

of the model. Consequently, this particular window size is solely used in the first stage, while the other stages refrain from its usage. This strategic decision contributes to managing computational demands while achieving high performance in object detection tasks.

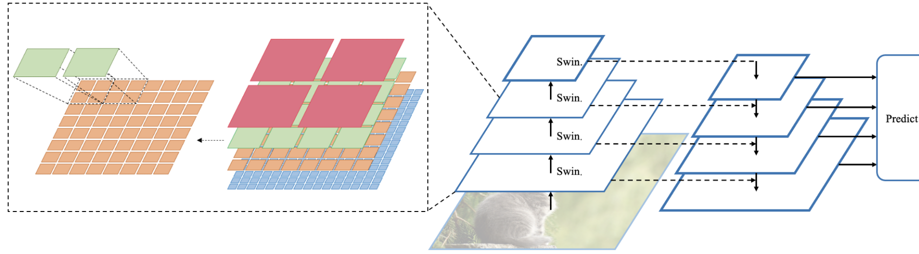

Figure 4.10: Pyramid Swin Transformer architecture

Each layer comprises two steps: one for window-based multi-head self-attention and another for shift window-based multi-head self-attention. Every layer is identical and includes two computations of multi-head self-attention, consistent with the Swin Transformer [80]. As shown in Figure 4.10, our model differs in that we split the attention process into smaller blocks. The number of different-sized windows in each layer follows a hierarchical progression from larger to smaller, promoting global connectivity. In each stage, the concluding layer (with the exception of the fourth stage) comprises a 2×2 window. The rationale behind this is that an optimal shift window-based multi-head self-attention is realized when the number of windows is 2×2. Under this configuration, all four windows are interconnected, thereby boosting the information exchange between windows and consequently augmenting global relevance. This strategic placement of windows ensures the balance between local feature refinement and global context understanding, thereby optimizing the overall performance of the model. It also brings out the inherent strength of the attention mechanism to make use of the positional relationships within the image.

Our proposed Pyramid Swin Transformer serves as an enhancement of the traditional Swin Transformer architecture, specifically tailored to boost efficiency and scalability within multi-task computer vision scenarios. Within this Pyramid Swin Transformer framework, we employ a new scheme to compute multi-head self-attention across windows of differing sizes. This approach facilitates the model's capacity to capture a blend of both localized and global image information.

Our Pyramid Swin Transformer layers process the input image and extract features at multiple scales, as same as the Swin Transformer. As shown

P.Swin-S	Size	Layers	Channel	Windows	Window size	Heads
Stage 1	64^2	3	96	$8^2, 4^2, 2^2$	$8^2, 16^2, 32^2$	3
Stage 2	32^2	2	192	$4^2, 2^2$	$8^2, 16^2$	6
Stage 3	16^2	2	384	$4^2, 2^2$	$4^2, 8^2$	12
Stage 4	8^2	1	768	1^2	8^2	24
P.Swin-R	**Size**	**Layers**	**Channel**	**Windows**	**Window size**	**Heads**
Stage 1	64^2	4	96	$16^2, 8^2, 4^2, 2^2$	$4^2, 8^2, 16^2, 32^2$	3
Stage 2	32^2	3	192	$8^2, 4^2, 2^2$	$4^2, 8^2, 16^2$	6
Stage 3	16^2	2	384	$4^2, 2^2$	$4^2, 8^2$	12
Stage 4	8^2	2	768	$2^2, 1^2$	$4^2, 8^2$	24
P.Swin-B	**Size**	**Layers**	**Channel**	**Windows**	**Window size**	**Heads**
Stage 1	64^2	4	128	$16^2, 8^2, 4^2, 2^2$	$4^2, 8^2, 16^2, 32^2$	4
Stage 2	32^2	3	256	$8^2, 4^2, 2^2$	$4^2, 8^2, 16^2$	8
Stage 3	16^2	3	512	$8^2, 4^2, 2^2$	$2^2, 4^2, 8^2$	16
Stage 4	8^2	2	1024	$2^2, 1^2$	$4^2, 8^2$	32
P.Swin-L	**Size**	**Layers**	**Channel**	**Windows**	**Window size**	**Heads**
Stage 1	64^2	4	192	$16^2, 8^2, 4^2, 2^2$	$4^2, 8^2, 16^2, 32^2$	6
Stage 2	32^2	3	384	$8^2, 4^2, 2^2$	$4^2, 8^2, 16^2$	12
Stage 3	16^2	3	768	$8^2, 4^2, 2^2$	$2^2, 4^2, 8^2$	24
Stage 4	8^2	2	1536	$2^2, 1^2$	$4^2, 8^2$	48

Table 4.1: Pyramid Swin Transformer Detailed architecture

in Figure 4.10, illustrates the Pyramid Swin Transformer using several times different-sized windows to compute window-based multi-head self-attention on the same scale of the feature map. The window size is set according to the feature map scale size. For large-scale feature maps, we use more times of W-MSA calculation, and also the window size is split into more different sizes. This approach, which involves conducting more multi-head self-attention calculations at larger scales, does slightly increase our computational load. However, it also aids in enhancing the model's comprehension and recognition of target objects. Moreover, large-scale features provide a wealth of contextual information, which is particularly beneficial for the attention mechanism. By leveraging the attention mechanism, the model can utilize this contextual information to capture long-distance dependencies between features, thereby enhancing the model's performance.

The detailed architecture is shown in Table 4.1, where the input image

size is 256×256. We refer to it as Pyramid Swin-R and the first stage has 4 layers, the second stage has 3 layers, the third stage has 2 layers, and the last stage has 2 layers. We also implement other frameworks, including Pyramid Swin-S, Pyramid Swin-B, and Pyramid Swin-L, which only differ from Pyramid Swin-R in the number of channels and layers. The details are as follows:

- Pyramid Swin-S: C = 96, layer = {3,2,2,1}
- Pyramid Swin-R: C = 96, layer = {4,3,2,2}
- Pyramid Swin-B: C = 128, layer = {4,3,3,2}
- Pyramid Swin-L: C = 192, layer = {4,3,3,2}

Here, C represents the number of channels in the first stage's layer, and each layer consists of two sub-multi-head self-attention calculations. The main idea behind the Pyramid Swin Transformer is to continuously add windows of various sizes to a uniform scale to improve their direct information contact. We utilize different window sizes in each layer, with the subsequent layer complementing the previous one. The issue with the Swin Transformer was the insufficient information interaction between windows at the low semantic level. Our improvements have enhanced the interaction between separate windows.

Suppose each window contains $2^i \times 2^i$ window sizes; on an image of size $h \times w$ feature map, the computational complexity of a global multi-head self-attention module and a window-based one is as follows:

$$\Omega(MSA) = 4hwC^2 + 2(hw)^2C$$
$$\Omega(W - MSA) = 4hwC^2 + (2)^{2i+1}hwC$$

Where C is a channel, the former complexity is quadratic to feature map size $h \times w$, while the latter mainly depends on the size of i, with $i \in 0, 5$. In our design, due to the large computation when $i = 5$, we attempt to minimize the occurrence of $i = 5$ in the entire framework. In fact, we only use $i = 5$ once at the first layer of the first stage. As window-based multi-head self-attention is scalable for $h \times w$, global self-attention computation is typically expensive. Therefore, window-based self-attention has great potential for reducing computation requirements. For self-attention computation, we follow [99, 54, 80] by including a relative position bias $\beta \in \mathbb{R}^{M^2 \times M^2}$ to each head:

$$Attention(Q, K, V) = SoftMax(QK^T/\sqrt{d} + \beta)V,$$

where $Q, K, V \in \mathbb{R}^{M^2 \times d}$ are the *query*, *key* and *value* matrices, d is the *query* and *key* dimension and M^2 is the window size.

4.3.1 Object Detection with FPN

In our Pyramid Swin Transformer architecture for object detection, we incorporate a Feature Pyramid Network (FPN) to enhance the model's ability to detect objects at different scales. The FPN was proposed by Lin et al. [75] as a top-down architecture with lateral connections to fuse high-level semantic information with low-level spatial details more effectively. This fusion enables the model to handle a wide range of object sizes and scales, making it particularly well-suited for object detection tasks. The FPN is constructed by taking the output feature maps from different layers of the base network and then combining them in a hierarchical manner. This is achieved by upsampling higher-level feature maps and adding them element-wise to lower-level feature maps, followed by a 1x1 convolution operation to reduce the number of channels. The resulting multi-scale feature maps are then fed into separate object detection heads to predict object classes and bounding box coordinates. By incorporating the FPN into our Pyramid Swin Transformer architecture, we can leverage the advantages of both the Swin Transformer's self-attention mechanism and the FPN's multi-scale object representation. This combination results in improved object detection performance across different object scales and varying levels of detail.

4.3.2 Semantic Segmentation Head

In order to provide a more detailed description of integrating the Pyramid Swin Transformer with UperNet for semantic segmentation, we first replace UperNet's[129] original backbone with our Pyramid Swin Transformer. This modification allows us to utilize the powerful feature extraction capabilities and hierarchical structure of the Pyramid Swin Transformer, which has been demonstrated to be effective for object detection tasks. The Pyramid Swin Transformer generates multi-scale feature maps at different stages of its architecture. These feature maps capture different levels of abstraction, ranging from low-level spatial details to high-level semantic information. By incorporating these feature maps into UperNet's pyramid pooling module(PPM head)[47], our combined model can effectively capture both local and global context information.

The pyramid pooling module[47], which serves to aggregate context from varying regions of an image by applying diverse scales of pooling, thereby encapsulating information that might be more apparent or impactful at different scales, in UperNet is designed to merge feature maps from different stages with varying levels of abstraction. It does so by applying pooling operations with different window sizes and strides, followed by bilinear up-

sampling to match the spatial dimensions. This process effectively fuses multi-scale features and allows the model to benefit from both high-level semantics and low-level spatial details. After the pyramid pooling module, the fused feature maps are processed through a series of convolutional layers and activation functions. These layers help refine the feature maps and extract more discriminative features for the semantic segmentation task. The refined feature maps are then used to generate per-pixel class probabilities, resulting in the final segmentation prediction. This output provides a high-resolution segmentation map with accurate and detailed class boundaries. By integrating the Pyramid Swin Transformer with UperNet[129], our approach effectively combines the strengths of both architectures. This fusion results in a powerful and efficient model for semantic segmentation, which has been shown to achieve state-of-the-art performance on the challenging ADE20K dataset [141].

4.3.3 Video Recognition Adaptations

To adapt the Pyramid Swin Transformer to video recognition tasks, as shown in Figure 4.1 (d model), we incorporate temporal information into the architecture by extending the input patches to include both spatial and temporal dimensions. Specifically, the input patches are formed by stacking consecutive frames from the input video along the temporal dimension, creating space-time video volumes. This approach allows the model to process the video input while preserving the temporal context. The attention mechanism in the transformer layers is also modified to consider temporal dependencies. This is achieved by incorporating temporal positional encodings, inspired by the work of Vaswani et al. [124], into the self-attention computation. Temporal positional encodings are added to the input embeddings, enabling the model to learn and exploit temporal relationships between video frames. This modification allows the Pyramid Swin Transformer to handle the temporal dynamics present in video data effectively.

In addition to the modifications to the attention mechanism, we also adjust the architecture of the Pyramid Swin Transformer to process the space-time video volumes more effectively. We introduce additional layers and connections specifically designed to capture the temporal information present in the video data, drawing inspiration from recent works on video recognition, such as TSN [127]. These additional components ensure that the model can effectively learn and exploit the relationships between video frames at different temporal scales. The output of the video recognition model is a sequence of class probabilities for each input frame. These probabilities represent the likelihood of the presence of specific objects or actions

in the corresponding video frames. The final video-level prediction can be generated by aggregating these frame-level probabilities using various strategies, such as averaging, max-pooling, or more sophisticated temporal pooling techniques, like the ones proposed in [36, 130].

By incorporating temporal information and modifying the attention mechanism, we successfully adapt the Pyramid Swin Transformer for video recognition tasks. This adaptation demonstrates the flexibility and potential of our proposed architecture for handling a wide range of computer vision tasks that involve both spatial and temporal dimensions.

4.3.4 Experiment and Result

Our experiments were conducted using the following hardware and software specifications, as shown in Table 4.2:

CPU	Intel(R) Xeon(R) Silver 4110
Memory	16G
GPU	NVIDIA Tesla V100 PCIe
GPU Memory	16G
Pytorch	1.8.1
CUDA	11.6
OS	Ubuntu 18.04

Table 4.2: Test Environment

We evaluate the performance of our proposed Pyramid Swin Transformer architecture on two benchmark datasets: ImageNet-1K for image classification [29] and COCO for object detection [77]. In the following sections, we present the results of our experiments and compare the performance of the Pyramid Swin Transformer to the state-of-the-art methods for these tasks. The primary metrics used for assessing our model encompass its accuracy, parameter count, and the number of Floating Point Operations (FLOPs). These three key indicators provide a holistic understanding of the model's performance, enabling an evaluation of the balance between its computational complexity and predictive accuracy. This analysis is critical for effective model deployment, particularly in resource-constrained environments where the trade-off between model complexity and prediction capability becomes crucial.

Image Classification on ImageNet

Settings. We benchmark the proposed Pyramid Swin Transformer on ImageNet-1K [29], which contains 1.28 million training images and 50k validation images from 1,000 classes. We report the single-crop top-1 accuracy, following the training methods used in Swin Transformer [80].
ImageNet-1K training. This setting mostly follows [122]. We employ an AdamW [61] optimizer for 300 epochs using a cosine decay learning rate scheduler, as in the Swin Transformer [80]. We include most of the augmentation and regularization strategies of [122] in training, except for repeated augmentation [51] and EMA [95]. This is in contrast to the situation where consistent augmentation is essential for maintaining ViT training [31].

Table 4.3.4 shows the results of our Pyramid Swin Transformer compared to state-of-the-art CNNs and other Transformer-based models. The models are divided into categories based on computation. Our Pyramid Swin Transformer achieves slightly better accuracy compared to state-of-the-art Convolution Nets and Vision Transformer models, such as RegNet [98], EfficientNet [121], CoAtNet [27], ViT [31], DeiT [122], MViTv2 [33], Swin [80], and SwinV2 [79].

Table 4.3.4, we make pre-train on ImageNet-1K. Our Pyramid Swin model is trained for 300 epochs without any external data or models. Our proposed design, the Pyramid Swin Transformer, has been shown to outperform several CNN systems on ImageNet, even when utilizing a small model (Pyramid Swin-S) and regular model (Pyramid Swin-R)4.3.4. However, our design does not exhibit significant advantages over Transformer systems in image classification, such as DeiT [122], MVit [33], and Swin Transformer [80]. Compared to the original Swin Transformer and Swin Transformer V2 [79], our improved version achieves greater accuracy while utilizing fewer parameters. For example, Pyramid Swin-R achieves the same accuracy as SwinV2-B with fewer parameters. On the regular-size model, Pyramid Swin-R outperforms Swin-B by +0.1% in accuracy, while on the large-size model, Pyramid Swin-L improves +0.8% over SwinV2-B. Our Pyramid Swin Transformer maintains accuracy while reducing computation compared to SwinV2-B. When compared to MVit with a (320×320) image size, our large model (Pyramid Swin-L) achieves higher accuracy but at a higher computational cost. On the other hand, our regular model (Pyramid Swin-R) achieves a +0.6% higher accuracy compared to MVit with a (224×224) image size.

Our architecture's impact on image classification tasks is rather nuanced, with the primary value proposition being a reduction in computational requirements while maintaining competitive accuracy. This is likely attributable to the design choices in our Pyramid Swin Transformer, which employs more

Method	Resolution	Params	FLOPs	Top-1 Acc.
RegNetY-4G [98]	224^2	21M	4G	80.0
RegNetY-8G [98]	224^2	39M	8G	81.7
RegNetY-16G [98]	224^2	84M	16G	82.9
EfficientNet-B5 [121]	456^2	30M	10G	83.6
EfficientNet-B6 [121]	528^2	43M	19G	84.0
EfficientNet-B7 [121]	600^2	66M	37G	84.4
CoAtNet-0 [27]	224^2	25M	4G	81.6
CoAtNet-1 [27]	224^2	42M	8G	83.3
CoAtNet-2 [27]	224^2	75M	16G	84.1
CoAtNet-3 [27]	224^2	168M	35G	**84.6**
Vit-B/16 [31]	384^2	86M	55G	77.9
Vit-L/16 [31]	384^2	307M	191G	76.5
DeiT-S [122]	224^2	22M	5G	79.8
DeiT-B [122]	224^2	86M	18G	81.8
DeiT-B [122]	384^2	86M	55G	83.1
MViTv2-T [74]	224^2	24M	5G	82.3
MViTv2-S [74]	224^2	35M	7G	83.6
MViTv2-B [74]	224^2	52M	11G	84.4
MViTv2-L [74]	224^2	218M	42G	85.3
Swin-T [80]	224^2	28M	5G	81.3
Swin-S [80]	224^2	50M	9G	83.0
Swin-B [80]	224^2	88M	15G	83.5
Swin-B [80]	384^2	88M	47G	84.5
SwinV2-T [79]	256^2	28M	7G	82.8
SwinV2-S [79]	266^2	50M	13G	84.1
SwinV2-B [79]	256^2	88M	22G	**84.6**
Pyramid Swin-S	256^2	64M	11G	**83.9**
Pyramid Swin-R	256^2	77M	14G	**84.6**
Pyramid Swin-B	256^2	123M	27G	**85.1**
Pyramid Swin-L	256^2	164M	39G	**85.4**

Table 4.3: Results on Imagenet Image Classification

frequent feature extraction processes on larger-scale feature maps and less so on smaller-scale ones, potentially influencing its performance in image

classification tasks. Despite these considerations, our Pyramid Swin Transformer exhibits competitive performance, outpacing several state-of-the-art Transformer models like SwinV2 and MVit-B-24. This advantage is particularly pronounced in achieving a desirable trade-off of higher accuracy with fewer parameters and reduced computational load, thereby highlighting the effectiveness and efficiency of our proposed architecture.

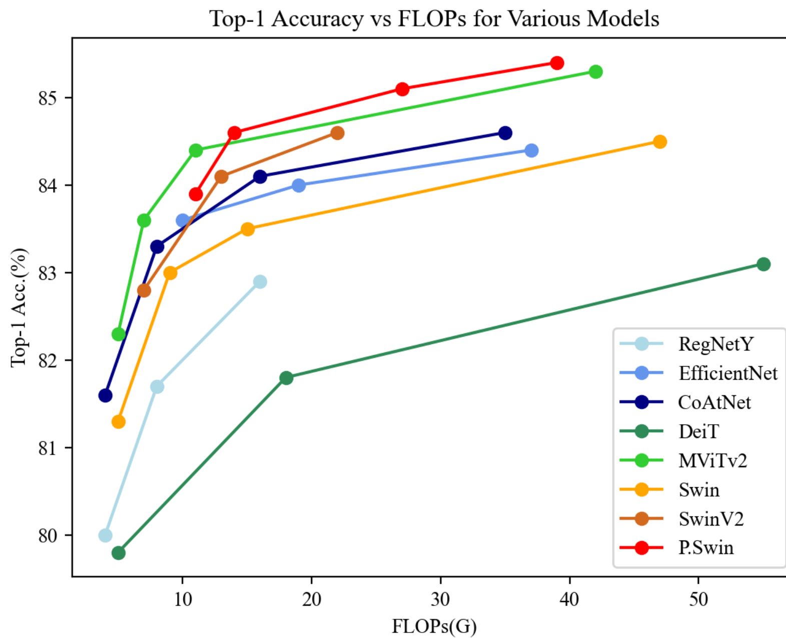

Figure 4.11: Top-1 Accuracy vs FLOPs for Various Models

Figure 4.11 presents a compelling depiction of the trade-off between model complexity(Number of Floating Point Operations(FLOPs)) and performance accuracy in several state-of-the-art models. Among these, our Pyramid Swin model emerges as a leading contender, delivering the highest accuracy among the models analyzed. However, it's worth noting that this superior performance comes with a relative increase in model complexity. Yet, the complexity level is within an acceptable range considering the improved accuracy, rendering the Pyramid Swin model a robust and efficient choice for high-demand, accuracy-centric applications.

An interesting trend emerges when we compare the performance of CNN-based networks with Transformer-based networks. It's evident that these different architectural paradigms have reached an almost identical level of complexity and accuracy, essentially leveling the playing field. Moreover, in

some cases, Transformer-based networks even outperform their CNN-based counterparts in terms of accuracy. This observation signals a promising shift in the landscape of computer vision. It suggests that the potential of Transformer-based models in this field is not only on par with but may exceed that of traditional CNNs. Given the inherent advantages of Transformer models, including their ability to model long-range dependencies and their flexibility in processing inputs of different sizes, their application in computer vision is likely to become even more prevalent.

Our Pyramid Swin model showcases the potential of Transformer-based models in achieving high accuracy for computer vision tasks, even with relatively higher computational complexity. Furthermore, the general trend towards equal performance between CNN and Transformer architectures suggests a bright future for Transformer models in the domain of computer vision. This trend is likely to fuel further research and development efforts aimed at refining and expanding the application of Transformer models for an even wider range of computer vision tasks.

Object Detection on COCO

We conduct object detection experiments on the Microsoft COCO [77] dataset. An ablation study is conducted using the validation set, and test-dev is used to report on a system-level comparison. We use standard Mask R-CNN [46] and Cascade Mask R-CNN [14] detection frameworks implemented in Detectron2. The backbone networks of the objects we compared are ResNet [48], PVT-S [128], ViL-S-RPB [137], and Swin [80]. For a fair comparison, we follow the same approach as Swin Transformer [80]. For these four frameworks, we utilize the same settings: multi-scale training [17, 118]. For Pyramid Swin, we take the backbone pre-trained from ImageNet-1K. The input sizes are set as $[64, 32, 16, 8]$ for the multi-scale four stages, consistent with the self-attention size used in ImageNet-1K pre-training.

When utilizing the Mask R-CNN framework , our Pyramid Swin Transformer achieved the highest accuracy on the regular-size model, outperforming other models in the comparison. Specifically, Pyramid Swin-R achieved an AP^{box} of 50.3, improving by +1.8 AP^{box} over Swin-B [80], while utilizing fewer parameters. Compared to ViT-B-RPB, our Pyramid Swin architecture demonstrated an advantage with a +0.7 box AP improvement. On the big-size and large-size model, Pyramid Swin-L achieved an AP^{box} of 51.1 and AP^{box} of 51.6, achieving better performance than Swin-B, despite utilizing more parameters. These results demonstrate the superior performance of our Pyramid Swin Transformer architecture when utilized with the Mask R-CNN framework for object detection tasks.

Model	AP^{box}	AP^{mask}	FLOPs	Params
Res50 [48]	41.0	37.1	260G	44M
Res101 [48]	42.8	38.5	336G	63M
X101-64 [63]	44.4	39.7	493G	101M
PVT-S [128]	43.0	39.9	245G	44M
PVT-M [128]	44.2	40.5	302G	64M
PVT-L [128]	44.5	40.7	364G	81M
ViL-S-RPB [137]	47.1	42.1	277 G	45M
ViL-M-RPB [137]	48.9	44.2	352G	60M
ViL-B-RPB [137]	49.6	44.5	384G	76M
MViTv2-T [74]	48.2	43.8	701G	76M
MViTv2-S [74]	49.9	45.1	748G	87M
MViTv2-B [74]	51.0	45.7	814G	103M
MViTv2-L [74]	51.8	46.2	1097G	238M
Swin-T [80]	46.0	41.6	264G	48M
Swin-S [80]	48.5	43.3	354G	69M
Swin-B [80]	48.5	43.4	496G	107M
Pyramid Swin-S	**49.9**	**44.2**	402G	78M
Pyramid Swin-R	**50.3**	**44.8**	463G	94M
Pyramid Swin-B	**51.1**	**45.3**	657G	137M
Pyramid Swin-L	**51.6**	**45.7**	864G	193M

Table 4.4: Results on COCO object detection with Mask R-CNN

Within the framework of Cascade Mask R-CNN, our Pyramid Swin Transformer has also demonstrated commendable performance outcomes, surpassing other models in the comparison. In particular, our Pyramid Swin-S outperformed all other models with an AP^{box} of 53.1, which is +1.3 AP^{box} higher than Swin-S [80]. However, MViTv2-S [74] achieves higher scores. Pyramid Swin-R achieved an AP^{box} of 53.6, surpassing Swin-B by +1.7 AP^{box} and MViTv2-B by +0.5 AP^{box}, while utilizing fewer parameters. In the large-size model, Pyramid Swin-L achieved an AP^{box} of 54.3. Our Pyramid Swin Transformer architecture outperformed other models when used in the Cascade Mask R-CNN framework for object detection tasks.

Our experimental results reveal that our architecture demonstrates the most robust performance in object detection tasks. This superiority is conjectured to stem from our model's design, which entails executing a greater

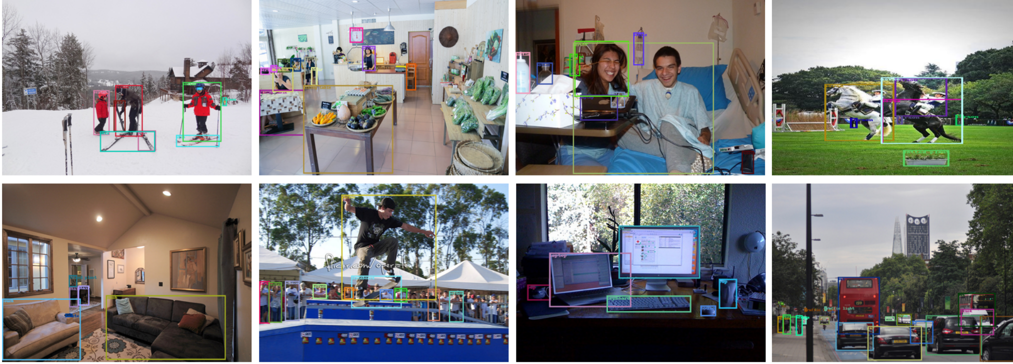

Figure 4.12: Object Detection on COCO

Model	AP^{box}	AP^{mask}	FLOPs	Params
Res50 [48]	46.3	40.1	739G	82M
Res101 [48]	47.7	40.8	819G	101M
MViTv2-T [74]	52.2	45.0	701G	76M
MViTv2-S [74]	53.2	46.0	748G	87M
MViTv2-B [74]	54.1	46.8	814G	103M
Swin-T [80]	50.5	43.7	745G	86M
Swin-S [80]	51.8	44.7	838G	107M
Swin-B [80]	51.9	45.0	982G	145M
Pyramid Swin-S	**53.1**	**45.9**	812G	114M
Pyramid Swin-R	**53.6**	**46.4**	902G	136M
Pyramid Swin-B	**54.0**	**47.1**	1247G	201M
Pyramid Swin-L	**54.3**	**47.5**	1567G	273M

Table 4.5: Results on COCO object detection with Cascade Mask R-CNN

number of window-based multi-headed self-attention computations on large-scale feature maps. This approach offers two advantages:

Firstly, by facilitating more attention computations on large feature maps, our model can better apprehend and utilize the global contextual information embedded within the image data. Such a capability is critical for object detection tasks as understanding the broader context of an image can often inform the model about the presence and positioning of objects within the scene.

Secondly, our Pyramid Swin Transformer's use of larger windows in the attention calculations further amplifies its performance. With larger windows, each self-attention operation has access to a richer set of local information within each window. This local richness can be particularly beneficial for object detection, as it enables the model to capture intricate details and relationships between neighboring pixels, thereby enhancing the model's ability to precisely locate and classify objects.

Ultimately, through these design choices that optimize both local detail extraction and global context understanding, our model has achieved state-of-the-art performance in object detection tasks, showing promise for further development and application.

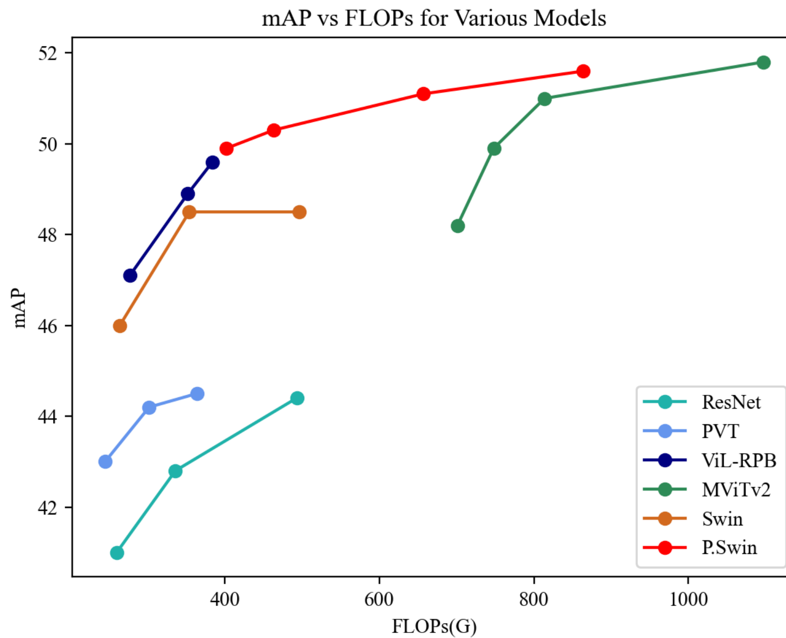

Figure 4.13: mAP vs FLOPs for Various Models(Mask R-CNN)

In our research, we illustrate the performance trade-off of various models in Figure 4.13, where we depict the relationship between Number of Floating Point Operations (FLOPs) and mean Average Precision (mAP) using the Mask R-CNN framework. In this context, it becomes clear that our novel architecture demonstrates a distinct advantage in terms of object detection capabilities. Observing Figure 4.13, one can ascertain that our model stands out in terms of achieving a finely balanced equilibrium between computational complexity, as indicated by FLOPs, and model accuracy, represented

by mAP. The superiority of our model becomes even more evident when considering that it effectively mitigates some of the inherent constraints of the original Swin Transformer architecture. These enhancements in our architecture not only bolster model performance but also contribute substantially towards optimizing computational resources. Hence, the evidence suggests that our model provides a promising avenue for the development of more efficient and accurate object detection systems.

Semantic Segmentation on ADE20K

We adopt the ADE20K dataset[141] for our semantic segmentation experiments, a widely used benchmark covering a diverse range of 150 semantic categories. The dataset comprises a total of 25,000 images, with 20,000 designated for training, 2,000 for validation, and 3,000 for testing.

Figure 4.14: Semantic Segmentation on ADE20K

For our base framework, we employ the efficient UperNet model[129], as implemented in the mmseg library[22]. This choice allows for a fair comparison with previous approaches in terms of both performance and efficiency. In our experiments, we train Pyramid Swin-S and Pyramid Swin-R models, following the same standard settings as those employed by prior methods. Specifically, we use an input size of 512×512 for both models, maintaining consistency with the input dimensions used in previous studies.

Our Pyramid Swin Transformer design achieves superior accuracy compared to Swin-B[80] and XCiT-S24/8[2] for the regular-size model (Pyramid Swin-R) in table 4.3.4, with a +0.4 mIoU improvement over Swin-B and a +1.4 mIoU improvement over XCiT-S24/8, all while maintaining a comparable number of parameters. Notably, further increasing the number of param-

Backbone	Method	mIoU	FLOPs	Params
ResNet-101 [48]	DANet [40]	47.1	1119G	69M
ResNet-101	OCRNet [135]	46.0	1249G	69M
ResNet-101	DNL [132]	49.6	384G	76M
ResNet-101	UperNet	44.9	1029G	86M
XCiT-S24/8 [2]	UperNet	47.1	-	74M
XCiT-M24/16 [2]	UperNet	45.9	-	109M
XCiT-M24/8 [2]	UperNet	46.9	-	109M
Swin-T [80]	UperNet	44.5	945G	60M
Swin-S [80]	UperNet	47.6	1038G	81M
Swin-B [80]	UperNet	48.1	1188G	121M
Pyramid Swin-S	UperNet	**47.9**	926G	92M
Pyramid Swin-R	UperNet	**48.5**	1091G	113M
Pyramid Swin-B	UperNet	**48.8**	1452G	161M
Pyramid Swin-L	UperNet	**49.0**	2036G	237M

Table 4.6: Results of ADE20K samantic segmentation with UperNet

eters within our architecture can lead to even better results. In particular, our big-size model attains a 48.8 mIoU, while our large-size model achieves a 49.0 mIoU. These results highlight the effectiveness of our Pyramid Swin Transformer approach in enhancing semantic segmentation performance.

While the Pyramid Swin Transformer demonstrates impressive performance on object detection tasks, its efficacy in semantic segmentation tasks may not be as robust. One possible explanation lies in the model's inherent feature extraction strategy. The Pyramid Swin Transformer tends to perform more multi-headed self-attention computations on large-scale feature maps, consequently emphasizing global context understanding. Conversely, it performs fewer computations on small-scale feature maps, which are crucial for semantic segmentation tasks that require high-resolution features for accurate pixel-level predictions. The balance between global and local information can also have an impact on semantic segmentation tasks. Despite the wealth of global information acquired from computations on large-scale feature maps, preserving and enhancing local details is equally critical for semantic segmentation. Therefore, our model may need further optimization to adapt to the task of semantic segmentation in order to achieve better results.

Video Recognition on Kinetics-400

We evaluate the performance of our Pyramid Swin Transformer on the Kinetics-400 dataset [60] (K400), which comprises approximately 240k training videos and 20k validation videos spanning 400 human action categories. Our training methodology follows that of [81]. Specifically, we employ an AdamW [61] optimizer for 30 epochs using a cosine decay learning rate scheduler and 2.5 epochs of linear warm-up, with a batch size of 64. As the backbone is initialized from a pre-trained model while the head is randomly initialized, we multiply the backbone learning rate by 0.1 to improve performance. The initial learning rates for the ImageNet pre-trained backbone and the randomly initialized head are set to 3e-5. To compute the final score, we take the average score overall views.

Model	Pre-train	Top-1 Acc.	FLOPs×views	Params
SlowFast 16×8 [36]	-	79.8	234×3×10	60M
X3D-XL [35]	-	79.1	48×3×10	11M
MoViNet-A6 [64]	-	81.5	386×1×1	31M
ViT-B-TimeSformer [8]	ImageNet-21K	80.7	2380×1×3	121M
ViT-B-VTN [89]	ImageNet-21K	78.6	4218×1×1	11M
ViViT-L/16×2 [3]	ImageNet-21K	80.6	1446×4×3	311M
ViViT-L/16×2 320	ImageNet-21K	81.3	3992×4×3	311M
Swin-T [80]	ImageNet-1K	78.8	88×4×3	28M
Swin-S [80]	ImageNet-1K	80.6	166×4×3	50M
Swin-B [80]	ImageNet-1K	80.6	282×4×3	88M
Swin-B	ImageNet-21K	82.7	282×4×3	88M
Pyramid Swin-S	ImageNet-1K	**80.8**	206×4×3	64M
Pyramid Swin-R	ImageNet-1K	**81.2**	261×4×3	77M
Pyramid Swin-R	ImageNet-21K	**83.4**	261×4×3	77M

Table 4.7: Results of Kinetics-400 video recognition.

Table 4.3.4 presents the results of Kinetics-400 video recognition, where we compare our Pyramid Swin Transformer models with other state-of-the-art models. SlowFast 16×8 +NL[36], X3D-X[35]L, and MoViNet-A6[64] are designed with different architectures. TimeSformer-L[8], ViT-B-VTN[89], ViViT-L/16×2[3], and ViViT-L/16×2 320[3] are pre-trained on ImageNet-21K. Swin-T, Swin-S, and Swin-B[80] are pre-trained on ImageNet-1K or ImageNet-21K.

Our Pyramid Swin Transformer models, including Pyramid Swin-S, Pyramid Swin-R, and Pyramid Swin-L, outperform Swin-B and other models in terms of accuracy with comparable or fewer parameters. Specifically, our Pyramid Swin-S achieves a top-1 accuracy of 80.8%, and Pyramid Swin-R achieves a top-1 accuracy of 81.2%, surpassing the other models in Table 4.3.4. Moreover, Pyramid Swin-R pre-trained on ImageNet-21K achieves the highest accuracy of 83.4% among all the models, highlighting the effectiveness of our Pyramid Swin Transformer models for video recognition tasks. In the task of video recognition, which fundamentally parallels image classification, the observed results could likely be attributed to the limited feature extraction conducted on small-scale feature maps. This leads to a considerable loss of detailed semantic information. Such an observation underscores the necessity of extracting more granular features for tasks like video recognition, where capturing temporal subtleties and intricate visual cues is imperative. Balancing the focus between larger context and finer details remains a key area of refinement for our Pyramid Swin Transformer in enhancing its performance across a broader spectrum of visual recognition tasks.

In conclusion, while our model demonstrates promising capabilities, it also highlights the necessity for continuous refinement and optimization. The efficacy of our Pyramid Swin Transformer is inherently tied to the careful tuning of model parameters, a process that should be intricately calibrated based on the specific requirements of different tasks at hand. It is imperative to not only take into account the intrinsic characteristics of the task, such as the scale of feature maps and the nature of attention required but also the computational constraints and efficiency demands. Through further iterative experimentation and fine-tuning of these variables, we believe that our model's performance can be considerably enhanced, paving the way towards superior results across a wide array of computer vision tasks.

4.4 High-speed Window-based Multi-head Self-attention

In this section, we expound on our high-velocity implementation aimed at expediting the window-based multi-head self-attention (W-MSA) process using CUDA. This is a central component of our proposed Pyramid Swin Transformer model. Given the substantial computational requirements of our Pyramid Swin Transformer model, the need for a mechanism to enhance computation speed is crucial. W-MSA, an efficient modification of the traditional self-attention mechanism, is specially crafted to address the computational

and memory constraints of standard Transformer models when working with large-scale images or extensive sequences. This variant not only preserves the powerful representational capacity of self-attention but also facilitates more efficient processing of large data, reinforcing the applicability and utility of our Pyramid Swin Transformer model. In the initial stages of our research, we identified that the Swin Transformer harbors a significant number of components that can be parallelized, particularly the foundational W-MSA. This realization implied that the use of GPUs could potentially enhance computational speed. The parallelization of these components, especially the W-MSA, is a key factor in our high-speed implementation. By leveraging the parallel processing capabilities of modern GPUs, we have managed to significantly reduce the computation time, thereby making our Pyramid Swin Transformer model more efficient and practical for real-world applications.

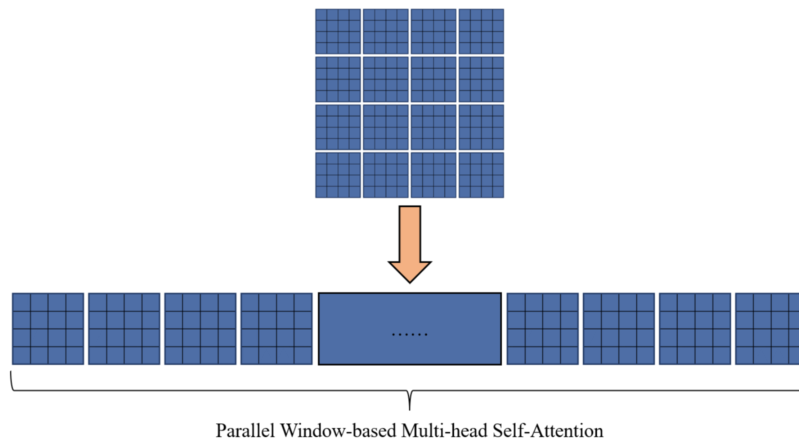

Figure 4.15: Parallel Window-based Multi-head Self-Attention

4.4.1 Our Method

In this section, we will describe our approach to accelerating window-based multi-headed self-attentive computation using CUDA, which lays the foundation for the enticing possibility of achieving an optimal trade-off between computational speed and model precision.

Matrix Multiplication with GPU

Matrix multiplication is at the core of numerous operations in deep learning, including the self-attention mechanism. Its efficient and scalable execution influences the overall performance of these models. Therefore, any

improvement in matrix multiplication can directly lead to an increase in the computational speed of the model. Recent developments in Graphics Processing Units (GPUs) have brought forth their potential to accelerate deep learning tasks. With their highly parallel structure, GPUs are particularly adept at processing these computationally intense tasks, thereby making them ideal for speeding up matrix multiplication. Leveraging the capabilities of GPUs, therefore, becomes an integral part of our method for accelerating the window-based multi-head self-attention mechanism. Our algorithm 3 optimizes the GPU memory bandwidth and computational resources, making it highly efficient for large matrix multiplications, a common operation in deep learning tasks.

Algorithm 3: Optimized Matrix Multiplication Kernel

```
   Data: Matrix A of size M x K, Matrix B of size K x N
   Result: Matrix C of size M x N
 1 Function matmul_kernel_opt(A, B, C, M, N, K):
 2     #define TILE_SIZE 32
 3     row ← blockIdx.y * blockDim.y + threadIdx.y
 4     col ← blockIdx.x * blockDim.x + threadIdx.x
 5     Allocate shared memory shared_A[TILE_SIZE][TILE_SIZE +
        1] and shared_B[TILE_SIZE][TILE_SIZE + 1]
 6     Initialize value ← 0
 7     for tileIdx in 0 to (K + TILE_SIZE - 1) / TILE_SIZE do
 8         if row < M and tileIdx * TILE_SIZE + threadIdx.x < K
            then
 9             shared_A[threadIdx.y][threadIdx.x] ← A[row * K +
                tileIdx * TILE_SIZE + threadIdx.x]
10         else
11             shared_A[threadIdx.y][threadIdx.x] ← 0
12         end
13         if col < N and tileIdx * TILE_SIZE + threadIdx.y < K then
14             shared_B[threadIdx.y][threadIdx.x] ← B[(tileIdx *
                TILE_SIZE + threadIdx.y) * N + col]
15         else
16             shared_B[threadIdx.y][threadIdx.x] ← 0
17         end
18         Synchronize threads
19         for e in 0 to TILE_SIZE do
20             value ← value + shared_A[threadIdx.y][e] *
                shared_B[e][threadIdx.x]
21         end
22         Synchronize threads
23     end
24     if row < M and col < N then
25         C[row * N + col] ← value
26     end
27 end
```

This is where the `TILE_SIZE` of 32 comes in. When performing tiled matrix multiplication, each thread block is responsible for computing a `TILE_SIZE` $\times$ `TILE_SIZE` output tile. If we make `TILE_SIZE` equal to the warp size, we can ensure that: Each thread block contains a whole number of warps, which helps the GPU efficiently schedule the execution of these

warps. The threads within a warp can efficiently cooperate to load data and perform computations because they are all working on adjacent elements of the matrices. By setting `TILE_SIZE` to the warp size of 32, we are aligning our computation with the underlying hardware architecture, which helps us maximize the utilization of the GPU and achieve high performance.

Parallelization Strategy

The computational process in both methods 4.16 we employ commences with the transformation of the input data into queries (Q), keys (K), and values (V) via a set of learnable weight matrices. Each transformation is executed through a matrix multiplication operation, a computation that is inherently parallelizable and can be efficiently performed on a GPU. Upon the generation of the Q, K, and V matrices, they undergo reshaping for subsequent processing. Specifically, these matrices are partitioned into several smaller matrices corresponding to different attention heads. Each head can independently process its respective portion of the input, enabling further parallelization. This "head-splitting" operation is executed in parallel by allocating different threads to different sections of the input matrices. Subsequent to the head-splitting, the scaled dot-product attention operation is conducted. The interaction between different positions within each window is computed. Notably, this computation is performed independently for each attention head and for each window in the input, facilitating large-scale parallelization. For each position, this involves a dot product operation between the corresponding Q and K vectors, a softmax operation, and then a weighted sum operation with the Value (V). Following the attention computation, the output from different attention heads is concatenated and transformed back to the original data shape, a process often referred to as "head merging". This operation is also highly parallelizable and is carried out by assigning different GPU threads to different sections of the output data. Ultimately, an output transformation is applied through a matrix multiplication with a learnable weight matrix. Similar to the input transformations, this operation is also highly parallelizable and can be efficiently executed on a GPU.

Nevertheless, the distinction lies in our adoption of thread blocks and grids of varying sizes in our implementation to yield superior results 4.16. The computational architecture is intrinsically three-dimensional, defined by the window size, the count of windows, and the number of attention heads (h). Depending on this three-dimensional structure, we can implement different parallelization strategies. Initially, we employed a 3D parallelization strategy in line with its construction.

In our 3D parallelization strategy, we theoretically size each block at

$window_size \times window_size \times window_number$. This means that each block is responsible for executing computations for several window-based multi-head self-attention. To put it simply, imagine that our data batch is a three-dimensional cube. Each block in our grid takes a smaller cube (or block) from this larger cube and processes it independently. This is where the power of parallel processing comes into play, as each of these blocks can be processed simultaneously by the GPU. Within each block, each thread is assigned the task of calculating the attention score for a single head of a single token within a single sequence in the batch.

However, due to the hardware limitations of the GPU, we need to adjust the thread block size during implementation. GPUs have a maximum limit on the number of threads per block (1024) and the total amount of shared memory per block. Therefore, we need to configure the thread block size to align with these limitations to ensure optimal performance and resource utilization. This is akin to adjusting the number of workers on the assembly line or the tasks assigned to each worker to ensure the factory runs efficiently. In practice, this means we might need to adjust the size of our blocks or the number of threads per block to fit within these constraints. For instance, if our window size is very large, we might need to use smaller blocks or fewer threads per block. Conversely, if our window size is small, we might be able to use larger blocks or more threads per block. This flexibility allows us to adapt our implementation to a wide variety of data sizes and structures, ensuring that our model remains efficient and effective across different tasks and datasets.

In our work, we have adopted a 2D parallelization 4.16 strategy in addition to the 3D parallelization approach, to fully leverage the computational prowess of modern GPUs. This strategy involves configuring the thread blocks to $(window_size, window_size)$ and grid size to $(window_number, head_number)$, where each block is responsible for executing computations for a window-based multi-head self-attention. Within each block, each thread is tasked with calculating the attention score for a single head of a single pixel within a single sequence in the batch.

This 2D parallelization strategy allows each thread to process all the features of a single pixel in parallel, thereby effectively utilizing the GPU's parallel processing capabilities. However, it's important to note that the thread block size can become quite large for larger window sizes or a greater number of windows, potentially exceeding the hardware limitations of certain GPUs. Therefore, this strategy is particularly well-suited to scenarios where the window size and the number of windows are not excessively large, and the GPU is capable of robust parallel processing. Our 2D parallelization strategy offers a flexible and efficient approach to processing large-scale image data in

the W-MSA model. By dynamically adjusting the thread block and grid sizes based on the input data, we can ensure optimal performance and resource utilization, making our model adaptable to a wide range of computer vision tasks with diverse data sets and input dimensions.

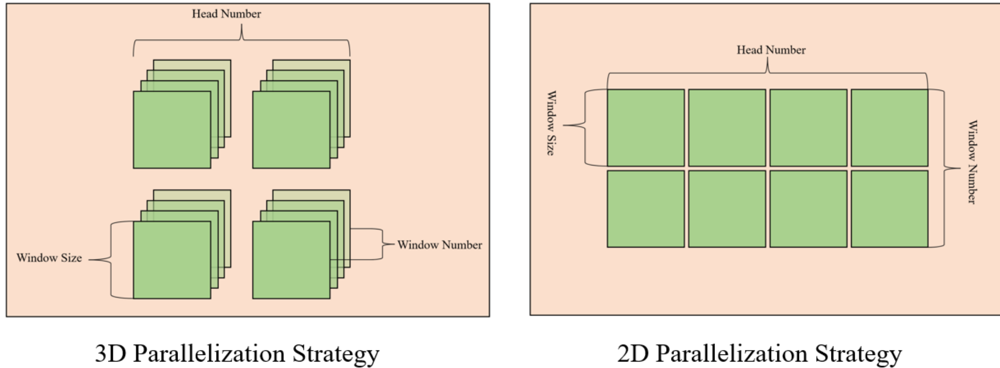

Figure 4.16: Parallelization Strategy

4.4.2 Result & Analysis

In this section, we present the results of our experiments and provide a detailed analysis of these outcomes. We evaluate our methods by focusing on three main aspects: matrix multiplication and window-based multi-head self-attention. Our experiments were conducted on a system equipped with powerful hardware and state-of-the-art software to ensure high-performance computations, as detailed in Table 4.8.

CPU	Intel(R) Xeon(R) Platinum 8255C CPU
Memory	32G
GPU	Nvidia GeForce RTX 2080Ti
GPU Memory	11G
Pytorch	1.8.1
CUDA	12.0
OS	Ubuntu 18.04

Table 4.8: Test Environment

Matrix Multiplication

Upon analysis of the provided test results as shown in figure 4.17, it is observed that for matrices with smaller dimensions ranging from 8×8 to 1024×1024, the custom kernel outperforms the cuBLAS library in terms of computation speed. The probable cause for this behavior can be attributed to the optimization approach of cuBLAS, which is specifically designed for handling larger matrices efficiently rather than smaller ones. The small-sized matrices do not fully utilize the inherent parallelism provided by cuBLAS, thereby leading to less efficient execution.

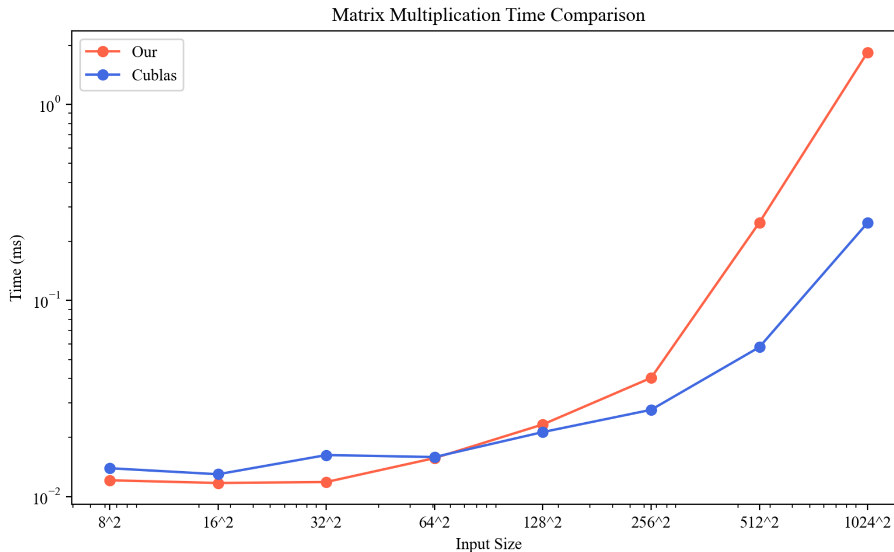

Figure 4.17: Comparison of Matrix Multiplication

When it comes to medium-sized matrices, specifically those with dimensions 448×448, the computation times for both the custom kernel and the cuBLAS library seem to converge. This is likely due to the reason that the custom kernel's effectiveness starts to decline with the growth of the matrix size, while the cuBLAS library begins to fully exploit its inherent parallelism to accelerate computation. For larger matrices, ranging from dimensions 512×512 to 1024×1024, the cuBLAS library surpasses the performance of the custom kernel. This superiority becomes increasingly pronounced as the matrix size grows. The custom kernel's performance decline is expected as the management of shared memory becomes more complex with larger matrices. In contrast, the cuBLAS library continues to benefit from its parallelization and optimization strategies, designed specifically for dealing with

large matrices. It is evident that for matrix multiplication tasks, the custom kernel demonstrates superior performance for smaller matrices, while cuBLAS is more efficient for larger ones. A mixed strategy can be employed in practice, switching between the custom kernel and cuBLAS based on the matrix size to achieve optimal performance.

The superior performance of our CUDA matrix multiplication implementation for smaller matrices can be attributed to its effective use of the tiling optimization technique and coalesced memory access pattern. The tiling technique involves dividing the input matrices into smaller 'tiles' or blocks, which are loaded into the GPU's faster-shared memory for computation. This method reduces the number of slower global memory accesses, providing a significant speedup, especially for smaller matrices where a large portion or the entire matrix can be loaded into shared memory at once. The coalesced memory access pattern, where threads in a block access consecutive memory addresses, further enhances memory access speed. This pattern allows these accesses to be combined into a single memory transaction, maximizing memory bandwidth utilization. However, it's important to note that these optimizations are relatively effective for smaller matrices. For larger matrices, libraries like cuBLAS, which are designed to handle a variety of matrix sizes and may employ more advanced algorithmic optimizations, can provide superior performance.

window-based multi-head self-attention

Figure 4.18 demonstrates the performance comparison when using a window size of 8×8. We also set batch size = 8, channel = 512 and head number = 8. Similarly, our window-based approach demonstrates superior performance across all tested settings.

As shown in Figure 4.18, we conducted another variation of the input size, this time focusing on an 8×8 window size. In line with the previous observations, the 2D Parallel implementation maintained its status as the superior performer. The efficiency gains over the PyTorch implementation were even more significant. Specifically, the 2D Parallel approach was approximately 5.85× faster than PyTorch(CPU) and 1.88× faster than PyTorch(GPU) with an input size of 16×16. More dramatically, with an input size of 64×64, it exhibited performance speeds 8.57× faster than PyTorch(CPU) and 1.88× faster than PyTorch(GPU). The 3D parallelization is about 1.65× faster than Pytorch (GPU) and about 7.35× faster than Pytorch (CPU) for an input size of 64×64, which also performs better than the Pytorch version.

These findings underscore the compelling advantages of our 2D Parallel implementation, particularly its ability to scale effectively with increasing

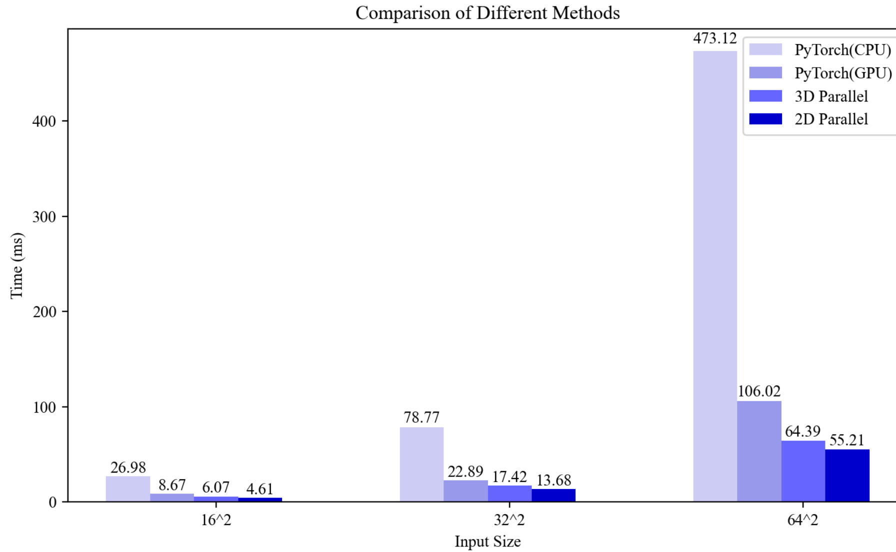

Figure 4.18: Performance comparison with 8×8 window size

input size. Our implementation capitalizes on the computational prowess of the GPU to a much greater extent than standard PyTorch implementations, yielding a performance boost that is quite noticeable. The improvements are substantial, with our version performing approximately two times faster than its PyTorch counterpart. These enhancements substantiate our approach's potential for practical applications, particularly in scenarios requiring high computational efficiency for window-based multi-head self-attention computations.

The parallelization strategy adopted for the Window-based Multi-head Self-Attention (W-MSA) model, which includes both 2D and 3D parallelization, provides a more efficient and optimized approach to GPU-based computations compared to the standard PyTorch implementation. By fully leveraging the parallel processing capabilities of modern GPUs and appropriately configuring thread blocks and grid sizes based on input data, this strategy ensures optimal utilization of GPU resources. It also incorporates coalesced memory access, significantly enhancing memory access speed and overall performance. Furthermore, by processing data in smaller chunks or 'windows', the strategy effectively reduces the memory footprint, leading to more efficient memory usage and faster execution times. Lastly, the use of custom CUDA kernels, specifically optimized for the computations involved in the W-MSA model, contributes to faster execution times. In summary, this strategy offers significant improvements in execution speed and overall performance,

demonstrating its effectiveness over the standard PyTorch implementation on GPU.w

The efficiency of our 2D parallelization strategy may be attributed to several factors that are intrinsic to the architecture of modern GPUs and the nature of the Window-based Multi-head Self-Attention (W-MSA) model.

- **Reduced Synchronization Overhead**: In a 3D parallelization strategy, the increased number of threads leads to a higher synchronization overhead. All threads must reach designated synchronization points before progressing, causing faster threads to idle while waiting for slower ones. This waiting period becomes more significant with a larger number of threads, as seen in 3D parallelization, thus contributing to the extension of the overall execution time and making the 3D scheme less efficient compared to the 2D approach.

- **Hardware Limitations**: GPU devices have a limit on the number of threads per block. For example, some devices may limit the maximum number of threads per block to 1024. If the product of your window size and window number exceeds this limit, 3D parallelization might not be able to utilize all threads effectively.

- **Flexibility**: The 2D parallelization strategy is more flexible and can easily adapt to different data sizes and GPU architectures. This is because the number of threads per block and the number of blocks per grid can be appropriately adjusted based on the input data size and the specific GPU architecture. This flexibility can lead to better utilization of the GPU's computational resources, resulting in faster execution times.

In conclusion, our 2D parallelization strategy for the W-MSA model effectively leverages the parallel processing capabilities of modern GPUs, while reducing synchronization overhead and so on. This results in a highly efficient and scalable solution for large-scale image processing tasks.

4.5 Conclusion

In conclusion, the Pyramid Swin Transformer is an effective architecture that achieves relatively remarkable performance in various computer vision tasks such as object detection, image classification, semantic segmentation, and video recognition. Our proposed design demonstrates consistent improvements over the original Swin Transformer and other state-of-the-art

models in terms of accuracy while making a better trade-off between computational complexity and performance. In object detection, the Pyramid Swin Transformer outperforms other models when utilized with Mask R-CNN and Cascade Mask R-CNN frameworks. For image classification, our architecture maintains competitive performance with other Transformer systems, even when using regular-sized models. In semantic segmentation, the Pyramid Swin Transformer achieves higher accuracy than competing models, showcasing its ability to capture rich semantic information. Lastly, in video recognition tasks, our architecture surpasses Swin-B and other state-of-the-art models in terms of accuracy, demonstrating its potential in spatiotemporal understanding.

The success of our Pyramid Swin Transformer can be attributed to its novel design, which utilizes different size windows on the same scale. This combination allows for efficient computation and more effective feature extraction, ultimately leading to improved performance across various computer vision tasks. The Pyramid Swin Transformer demonstrates the versatility and effectiveness of transformer-based architectures in the computer vision domain, and it has the potential to serve as a solid foundation for future research and applications in this field.

Moreover, our research has led to advancements in the acceleration of window-based multi-head self-attention (W-MSA). By harnessing the power of CUDA, we have increased the speed of W-MSA computations. Our initial breakthrough involved the enhancement of matrix multiplication, tailoring it for smaller sizes, thereby optimizing its compatibility with multi-head self-attention. Following this, we devised a 3D parallelization strategy and a 2D parallelization strategy that proved to be effective, resulting in a speed-up compared to the original W-MSA implementation. This paves the way towards the promising potential of a superior trade-off between processing speed and model accuracy. This cumulative progress lays a robust foundation for future exploration and signifies a leap in the optimization of the W-MSA mechanism.

To sum up, our comprehensive investigation and innovation revolving around the Pyramid Swin Transformer, as well as our advancements in accelerating W-MSA, highlight the potential and flexibility of transformer-based models in the arena of computer vision. By merging the strengths of the pyramid vision transformers and Swin Transformer architecture, we have devised an efficient, accurate, and high-performing model that sets a promising precedent for future research. Furthermore, our exploration of the acceleration of W-MSA computation marks a considerable stride in the domain. We've pushed the boundaries of computational efficiency with CUDA, paving the way for faster, yet reliable Swin Transformer-based models. This not only

amplifies the feasibility of applying these models in real-world tasks but also inspires further enhancements and applications in this continually evolving field.

Chapter 5

Conclusion & Future Work

5.1 Conclusion

Throughout this dissertation, we have taken an extensive exploration into GPU-based acceleration of computer vision task models, focusing on three main avenues of research: Single Shot MultiBox Detector (SSD) optimization using CUDA, the development and application of the Pyramid Swin Transformer, and the high-speed implementation of window-based multi-head self-attention mechanisms,

In the third chapter, we analyzed the potential of applying CUDA to speed up the SSD, a widely used object detection model. After a comprehensive investigation into the computational requirements of the SSD, we identified key areas where GPU-accelerated computation could enhance performance. Using CUDA kernels, we transferred the computational burden to the GPU, thereby enhancing the overall runtime of the model. The results were relatively notable, showing a substantial performance boost of about 22.53% faster than the original version which underscored the viability of applying GPUs to accelerate object detection frameworks and making an effective trade-off between speed and accuracy. However, we believe there are still opportunities for further acceleration, such as exploring CUDA's advanced capabilities, including concurrent kernel execution and dynamic parallelism. These techniques hold the potential to unlock higher levels of performance and will be a focal point of our future work.

In the fourth chapter, we focused on the Transformer-based models, specifically the Swin Transformer, which has demonstrated impressive performance in computer vision tasks. Recognizing the potential of this model and identifying its limitations, we developed an innovative solution: the Pyramid Swin Transformer, achieving a better balance between precision and com-

putational complexity. By incorporating more diverse window sizes, we increased the model's adaptability and performance, realizing a better trade-off between computational complexity and accuracy. Our architecture has been extensively evaluated on multiple benchmarks, including achieving 85.4% top-1 accuracy on ImageNet for image classification, 51.6 AP^{box} with Mask R-CNN and 54.3 AP^{box} with Cascade Mask R-CNN on COCO for object detection, 49.0 mIoU on ADE20K for semantic segmentation, and 83.4 % top-1 accuracy on Kinetics-400 for video recognition. This model adeptly captures local and global contextual information, providing improved performance over existing state-of-the-art models. We also introduced the high-speed implementation of the window-based multi-head self-attention(W-MSA) mechanism for GPU acceleration. To optimize this key component of the window-based multi-head self-attention, we first focus on the matrix multiplication operation, which is the basement of W-MSA. Our method is better than cuBLAS when the matrix size is relatively small. We proposed a parallel processing approach of W-MSA that outperformed the original PyTorch implementation. We foresee that our acceleration approach will enhance the speed of both training and detection within our model architecture, thus fostering the potential to strike a superior trade-off between precision and velocity.

In conclusion, this dissertation has made several important contributions to the field of computer vision by demonstrating novel and efficient methods for GPU-based acceleration and optimization methods. We believe that these contributions not only provide practical solutions to current challenges in the field but also open up new opportunities and directions for future work. However, the rapid advancement of computer vision and machine learning technologies means there is always more to do. With each contribution, we strive not just to advance the state of the art but also to provide a solid foundation for further exploration and development in this dynamic and evolving field.

5.2 Future Work

In the future, one promising direction is to explore more advanced CUDA optimization techniques for object detection models. While our research has shown the potential for substantial performance improvements by optimizing the Single Shot MultiBox Detector (SSD) with CUDA, there are additional CUDA techniques that could further enhance performance. These advanced techniques include concurrent kernel execution, dynamic parallelism, and texture memory optimization. The integration and adaptation of these tech-

niques to SSD and other object detection models would be a challenging yet worthwhile endeavor, potentially leading to substantial performance gains. Porting our approach to some other detection architectures will also be one of our future topics, even some other computer vision task architectures, such as semantic segmentation, etc.

Moreover, the Pyramid Swin Transformer proposed in our research opens up a myriad of possibilities for Transformer architectures in computer vision. Future work could extend these models for other computer vision tasks not covered in this dissertation, such as depth estimation, optical flow prediction, and 3D object detection. It would be intriguing to observe the potential of these models in these tasks and to make adaptations and refinements as necessary to excel in these new challenges. In terms of the high-speed window-based multi-head self-attention mechanism, the possibility of integrating the entire approach into a full-fledged Swin Transformer model or our Pyramid Swin Transformer is an exciting prospect. This integration would enable a comprehensive evaluation of the impact on end-to-end tasks, including accuracy and complete training and inference time.

Finally, expanding upon the future work highlighted above, there's another promising and practical direction of research: the adaptation and testing of our methods on smaller-scale GPUs for real-world applications. As machine learning models are being increasingly deployed in smaller, embedded devices for on-the-edge processing, optimizing our work for these less powerful GPUs becomes a critical endeavor. This line of research involves efficiently porting our models to smaller, possibly resource-constrained, GPU architectures. This will likely necessitate further model optimizations to maintain the balance between accuracy and resource efficiency, and potentially introduce the need for techniques like model quantization, pruning, or distillation. Moreover, this work opens up a myriad of opportunities for real-time, in-the-field applications of object detection and other vision tasks. This could range from real-time surveillance systems, and autonomous vehicles, to handheld devices with visual recognition capabilities, among others. The practical application of our research for on-device, real-time computer vision tasks on smaller GPUs represents a rich vein of future research. This direction not only presents a host of technical challenges but also aligns with the current trend towards edge computing and the Internet of Things (IoT), making it an exciting prospect for future work.

In conclusion, the future of GPU-based acceleration for computer vision task models is promising. The research presented in this dissertation serves as a foundation for future explorations and innovations in this field. As technology advances and computational demands continue to increase, the techniques and insights gained from this research will be instrumental in the

quest for more efficient and effective computer vision models. The road ahead is teeming with challenges and opportunities, and we eagerly anticipate the continued evolution of this field.

Bibliography

[1] Edward H Adelson, Charles H Anderson, James R Bergen, Peter J Burt, and Joan M Ogden. Pyramid methods in image processing. *RCA engineer*, 29(6):33–41, 1984.

[2] Alaaeldin Ali, Hugo Touvron, Mathilde Caron, Piotr Bojanowski, Matthijs Douze, Armand Joulin, Ivan Laptev, Natalia Neverova, Gabriel Synnaeve, Jakob Verbeek, et al. Xcit: Cross-covariance image transformers. *Advances in neural information processing systems*, 34:20014–20027, 2021.

[3] Anurag Arnab, Mostafa Dehghani, Georg Heigold, Chen Sun, Mario Lučić, and Cordelia Schmid. Vivit: A video vision transformer. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6836–6846, 2021.

[4] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[5] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017.

[6] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Computer vision and image understanding*, 110(3):346–359, 2008.

[7] Josh Beal, Eric Kim, Eric Tzeng, Dong Huk Park, Andrew Zhai, and Dmitry Kislyuk. Toward transformer-based object detection. *arXiv preprint arXiv:2012.09958*, 2020.

[8] Gedas Bertasius, Heng Wang, and Lorenzo Torresani. Is space-time attention all you need for video understanding? In *ICML*, volume 2, page 4, 2021.

[9] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

[10] Navaneeth Bodla, Bharat Singh, Rama Chellappa, and Larry S Davis. Soft-nms–improving object detection with one line of code. In *Proceedings of the IEEE international conference on computer vision*, pages 5561–5569, 2017.

[11] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of the 19th international conference on computational statistics*, pages 177–186. Springer, 2010.

[12] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11):1222–1239, 2001.

[13] Leo Breiman. Random forests. *Machine learning*, 45:5–32, 2001.

[14] Zhaowei Cai and Nuno Vasconcelos. Cascade r-cnn: Delving into high quality object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6154–6162, 2018.

[15] Ph.D. Cameron R. Wolfe. Using transformers for computer vision. https://towardsdatascience.com/using-transformers-for-computer-vision-6f764c5a078b, 2022.

[16] John Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679–698, 1986.

[17] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I 16*, pages 213–229. Springer, 2020.

[18] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2017.

[19] Bowen Cheng, Bin Xiao, Jingdong Wang, Honghui Shi, Thomas S Huang, and Lei Zhang. Higherhrnet: Scale-aware representation learning for bottom-up human pose estimation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5386–5395, 2020.

[20] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014.

[21] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[22] MMSegmentation Contributors. MMSegmentation: Openmmlab semantic segmentation toolbox and benchmark. `https://github.com/open-mmlab/mmsegmentation`, 2020.

[23] Shane Cook. *CUDA programming: a developer's guide to parallel computing with GPUs*. Newnes, 2012.

[24] James W Cooley and John W Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90):297–301, 1965.

[25] A Criminisi, J Shotton, and E Konukoglu. Decision forests for classification, regression, density estimation, manifold learning and semi-supervised learning [internet]. *Microsoft Research*, 2011.

[26] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-fcn: Object detection via region-based fully convolutional networks. *Advances in neural information processing systems*, 29, 2016.

[27] Zihang Dai, Hanxiao Liu, Quoc V Le, and Mingxing Tan. Coatnet: Marrying convolution and attention for all data sizes. *Advances in Neural Information Processing Systems*, 34:3965–3977, 2021.

[28] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 1, pages 886–893. Ieee, 2005.

[29] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[30] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[31] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[32] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88:303–338, 2010.

[33] Haoqi Fan, Bo Xiong, Karttikeya Mangalam, Yanghao Li, Zhicheng Yan, Jitendra Malik, and Christoph Feichtenhofer. Multiscale vision transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6824–6835, 2021.

[34] Neetu Faujdar and Satya Prakash Ghrera. Performance evaluation of merge and quick sort using gpu computing with cuda. *International Journal of Applied Engineering Research (IJAER)*, 10(18):39315–9, 2015.

[35] Christoph Feichtenhofer. X3d: Expanding architectures for efficient video recognition. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 203–213, 2020.

[36] Christoph Feichtenhofer, Haoqi Fan, Jitendra Malik, and Kaiming He. Slowfast networks for video recognition. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6202–6211, 2019.

[37] Pedro F Felzenszwalb, Ross B Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. *IEEE transactions on pattern analysis and machine intelligence*, 32(9):1627–1645, 2009.

[38] Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, and Dinani Amorim. Do we need hundreds of classifiers to solve real world classification problems? *The journal of machine learning research*, 15(1):3133–3181, 2014.

[39] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[40] Jun Fu, Jing Liu, Jie Jiang, Yong Li, Yongjun Bao, and Hanqing Lu. Scene segmentation with dual relation-aware attention network. *IEEE Transactions on Neural Networks and Learning Systems*, 32:2547–2560, 2020.

[41] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[42] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[43] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[44] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[45] Robert M. Haralick and Linda G. Shapiro. Image segmentation techniques. *Computer Vision, Graphics, and Image Processing*, 29(1):100–132, 1985.

[46] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[47] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(9):1904–1916, 2015.

[48] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[49] Yihui He, Chenchen Zhu, Jianren Wang, Marios Savvides, and Xiangyu Zhang. Bounding box regression with uncertainty for accurate object detection. In *Proceedings of the ieee/cvf conference on computer vision and pattern recognition*, pages 2888–2897, 2019.

[50] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, et al. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

[51] Elad Hoffer, Tal Ben-Nun, Itay Hubara, Niv Giladi, Torsten Hoefler, and Daniel Soudry. Augment your batch: Improving generalization through instance repetition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8129–8138, 2020.

[52] Jan Hosang, Rodrigo Benenson, and Bernt Schiele. Learning non-maximum suppression. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4507–4515, 2017.

[53] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2017.

[54] Han Hu, Zheng Zhang, Zhenda Xie, and Stephen Lin. Local relation networks for image recognition. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 3463–3472, 2019.

[55] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[56] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7310–7311, 2017.

[57] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[58] Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun. What is the best multi-stage architecture for object recognition? In *2009 IEEE 12th international conference on computer vision*, pages 2146–2153. IEEE, 2009.

[59] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014.

[60] Will Kay, Joao Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, et al. The kinetics human action video dataset. *arXiv preprint arXiv:1705.06950*, 2017.

[61] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[62] Alexander Kirillov, Ross Girshick, Kaiming He, and Piotr Dollár. Panoptic feature pyramid networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 6399–6408, 2019.

[63] Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Joan Puigcerver, Jessica Yung, Sylvain Gelly, and Neil Houlsby. Big transfer (bit): General visual representation learning. In *European Conference on Computer Vision*, 2019.

[64] Dan Kondratyuk, Liangzhe Yuan, Yandong Li, Li Zhang, Mingxing Tan, Matthew Brown, and Boqing Gong. Movinets: Mobile video networks for efficient video recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16020–16030, 2021.

[65] Oliver Kramer and Oliver Kramer. K-nearest neighbors. *Dimensionality reduction with unsupervised nearest neighbors*, pages 13–23, 2013.

[66] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[67] Mohan Pawan Kumar and Andrew Zisserman. Object detection using discriminatively trained part-based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9):1627–1645, 2005.

[68] Andrew Lavin and Scott Gray. Fast algorithms for convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4013–4021, 2016.

[69] Hei Law and Jia Deng. Cornernet: Detecting objects as paired keypoints. In *Proceedings of the European conference on computer vision (ECCV)*, pages 765–781, 2018.

[70] Yann LeCun and Yoshua Bengio. Convolutional networks for images, speech, and time-series. *Handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[71] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86:2278–2324, 1998.

[72] Yann LeCun, Koray Kavukcuoglu, and Clément Farabet. Convolutional networks and applications in vision. In *Proceedings of 2010 IEEE international symposium on circuits and systems*, pages 253–256. IEEE, 2010.

[73] Congcong Li. High quality, fast, modular reference implementation of SSD in PyTorch. `https://github.com/lufficc/SSD`, 2018.

[74] Yanghao Li, Chao-Yuan Wu, Haoqi Fan, Karttikeya Mangalam, Bo Xiong, Jitendra Malik, and Christoph Feichtenhofer. Mvitv2: Improved multiscale vision transformers for classification and detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4804–4814, 2022.

[75] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017.

[76] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[77] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pages 740–755. Springer, 2014.

[78] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[79] Ze Liu, Han Hu, Yutong Lin, Zhuliang Yao, Zhenda Xie, Yixuan Wei, Jia Ning, Yue Cao, Zheng Zhang, Li Dong, et al. Swin transformer v2: Scaling up capacity and resolution. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12009–12019, 2022.

[80] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. *arXiv preprint arXiv:2103.14030*, 2021.

[81] Ze Liu, Jia Ning, Yue Cao, Yixuan Wei, Zheng Zhang, Stephen Lin, and Han Hu. Video swin transformer. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 3202–3211, 2022.

[82] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[83] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[84] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Atlanta, Georgia, USA, 2013.

[85] Tomasz Malisiewicz, Abhinav Gupta, and Alexei A Efros. Ensemble of exemplar-svms for object detection and beyond. In *2011 International conference on computer vision*, pages 89–96. IEEE, 2011.

[86] David Marr and Ellen Hildreth. Theory of edge detection. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, 207(1167):187–217, 1980.

[87] Sanyam Mehta, Arindam Misra, Ayush Singhal, Praveen Kumar, and Ankush Mittal. A high-performance parallel implementation of sum

of absolute differences algorithm for motion estimation using cuda. In *HiPC Conf*, volume 2, page 6, 2010.

[88] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[89] Daniel Neimark, Omri Bar, Maya Zohar, and Dotan Asselmann. Video transformer network. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3163–3172, 2021.

[90] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable parallel programming with cuda. *Queue*, 6(2):40–53, 2008.

[91] NVIDIA. Cuda toolkit documentation. `https://docs.nvidia.com/cuda`, 2023.

[92] David Oro, Carles Fernández, Xavier Martorell, and Javier Hernando. Work-efficient parallel non-maximum suppression for embedded GPU architectures. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1026–1030. IEEE, 2016.

[93] John D Owens, Mike Houston, David Luebke, Simon Green, John E Stone, and James C Phillips. Gpu computing. *Proceedings of the IEEE*, 96(5):879–899, 2008.

[94] John D Owens, David Luebke, Naga K Govindaraju, Mark Harris, Jens Krüger, Aaron E Lefohn, and Tim Purcell. A survey of general-purpose computation on graphics hardware. *Computer graphics forum*, 26(1):80–113, 2007.

[95] Boris T Polyak and Anatoli B Juditsky. Acceleration of stochastic approximation by averaging. *SIAM journal on control and optimization*, 30:838–855, 1992.

[96] Xipeng Qiu, Tianxiang Sun, Yige Xu, Yunfan Shao, Ning Dai, and Xuanjing Huang. Pre-trained models for natural language processing: A survey. *Science China Technological Sciences*, 63(10):1872–1897, 2020.

[97] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[98] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10428–10436, 2020.

[99] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21:5485–5551, 2020.

[100] Rajat Raina, Anand Madhavan, and Andrew Y Ng. Large-scale deep unsupervised learning using graphics processors. In *Proceedings of the 26th annual international conference on machine learning*, pages 873–880, 2009.

[101] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

[102] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[103] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.

[104] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[105] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems*, pages 91–99, 2015.

[106] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. " grabcut" interactive foreground extraction using iterated graph cuts. *ACM transactions on graphics (TOG)*, 23(3):309–314, 2004.

[107] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[108] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115:211–252, 2015.

[109] Shane Ryoo, Christopher I Rodrigues, Sam S Baghsorkhi, Stephen S Stone, David B Kirk, and Wen-mei W Hwu. Optimization principles and application performance evaluation of a multithreaded gpu using cuda. *ACM SIGPLAN Notices*, 43(6):73–82, 2008.

[110] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. *Advances in neural information processing systems*, 30, 2017.

[111] Rafael Sachetto Oliveira, Bernardo Martins Rocha, Ronan Mendonça Amorim, Fernando Otaviano Campos, Wagner Meira, Elson Magalhães Toledo, and Rodrigo Weber dos Santos. Comparing cuda, opencl and opengl implementations of the cardiac monodomain equations. In *Parallel Processing and Applied Mathematics: 9th International Conference, PPAM 2011, Torun, Poland, September 11-14, 2011. Revised Selected Papers, Part II 9*, pages 111–120. Springer, 2012.

[112] Matsuoka Satoshi, Endo Toshio, Nukada Akira, Miura Shinichi, Nomura Akihiro, Sato Hitoshi, Jitsumoto Hideyuki, and Drozd Aleksandr. Overview of tsubame3. 0, green cloud supercomputer for convergence of hpc, ai and big-data. *Tsubame ESJ.: e-science journal/Tsubame e-science journal*, 16:20–27, 2017.

[113] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

[114] Jamie Shotton, John Winn, Carsten Rother, and Antonio Criminisi. Textonboost for image understanding: Multi-class object recognition and segmentation by jointly modeling texture, layout, and context. *International journal of computer vision*, 81:2–23, 2009.

[115] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[116] Irwin Sobel and Gary Feldman. Camera models and machine perception. *PhD thesis, Stanford University*, 1968.

[117] Robin Strudel, Ricardo Garcia, Ivan Laptev, and Cordelia Schmid. Segmenter: Transformer for semantic segmentation. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 7262–7272, 2021.

[118] Peize Sun, Rufeng Zhang, Yi Jiang, Tao Kong, Chenfeng Xu, Wei Zhan, Masayoshi Tomizuka, Lei Li, Zehuan Yuan, Changhu Wang, et al. Sparse r-cnn: End-to-end object detection with learnable proposals. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 14454–14463, 2021.

[119] Matthews Suzanne J, Newhall Tia, and Webb Kevin C. *Dive Into Systems: A Gentle Introduction to Computer Systems*. No Starch Press, 020.

[120] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[121] Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019.

[122] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International conference on machine learning*, pages 10347–10357. PMLR, 2021.

[123] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 4489–4497, 2015.

[124] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[125] Paul Viola and Michael J. Jones. Rapid object detection using a boosted cascade of simple features. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages I–I, 2001.

[126] Vasily Volkov and James W Demmel. Benchmarking gpus to tune dense linear algebra. In *2008 SC-International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–11. IEEE, 2008.

[127] Limin Wang, Yuanjun Xiong, Zhe Wang, Yu Qiao, Dahua Lin, Xiaoou Tang, and Luc Van Gool. Temporal segment networks: Towards good practices for deep action recognition. In *European conference on computer vision*, pages 20–36. Springer, 2016.

[128] Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 568–578, 2021.

[129] Tete Xiao, Yingcheng Liu, Bolei Zhou, Yuning Jiang, and Jian Sun. Unified perceptual parsing for scene understanding. In *Proceedings of the European conference on computer vision (ECCV)*, pages 418–434, 2018.

[130] Saining Xie, Chen Sun, Jonathan Huang, Zhuowen Tu, and Kevin Murphy. Rethinking spatiotemporal feature learning for video understanding. *arXiv preprint arXiv:1712.04851*, 1:5, 2017.

[131] Yuwen Xiong, Renjie Liao, Hengshuang Zhao, Rui Hu, Min Bai, Ersin Yumer, and Raquel Urtasun. Upsnet: A unified panoptic segmentation network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8818–8826, 2019.

[132] Minghao Yin, Zhuliang Yao, Yue Cao, Xiu Li, Zheng Zhang, Stephen Lin, and Han Hu. Disentangled non-local neural networks. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XV 16*, pages 191–207. Springer, 2020.

[133] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[134] Li Yuan, Yunpeng Chen, Tao Wang, Weihao Yu, Yujun Shi, Zi-Hang Jiang, Francis EH Tay, Jiashi Feng, and Shuicheng Yan. Tokens-to-token vit: Training vision transformers from scratch on imagenet. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 558–567, 2021.

[135] Yuhui Yuan, Xilin Chen, and Jingdong Wang. Object-contextual representations for semantic segmentation. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part VI 16*, pages 173–190. Springer, 2020.

[136] Aston Zhang, Zachary C Lipton, Mu Li, and Alexander J Smola. Dive into deep learning. *arXiv preprint arXiv:2106.11342*, 2021.

[137] Pengchuan Zhang, Xiyang Dai, Jianwei Yang, Bin Xiao, Lu Yuan, Lei Zhang, and Jianfeng Gao. Multi-scale vision longformer: A new vision transformer for high-resolution image encoding. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 2998–3008, 2021.

[138] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2881–2890, 2017.

[139] Qijie Zhao, Tao Sheng, Yongtao Wang, Zhi Tang, Ying Chen, Ling Cai, and Haibin Ling. M2det: A single-shot object detector based on multi-level feature pyramid network. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 9259–9266, 2019.

[140] Sixiao Zheng, Jiachen Lu, Hengshuang Zhao, Xiatian Zhu, Zekun Luo, Yabiao Wang, Yanwei Fu, Jianfeng Feng, Tao Xiang, Philip HS Torr, et al. Rethinking semantic segmentation from a sequence-to-sequence perspective with transformers. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 6881–6890, 2021.

[141] Bolei Zhou, Hang Zhao, Xavier Puig, Tete Xiao, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Semantic understanding of scenes through the ade20k dataset. *International Journal of Computer Vision*, 127:302–321, 2019.

[142] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. Objects as points. *arXiv preprint arXiv:1904.07850*, 2019.

Appendix A

Publish List

[1] Chenyu Wang, Toshi Endo, Takahiro Hirofuchi, and Tsutomu Ikegami. Speed-up single shot detector on GPU with CUDA. *In 2022 23rd ACIS International Summer Virtual Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD-Summer)*, pages 36–41, 2022.

[2] Chenyu Wang, Toshio Endo, Takahiro Hirofuchi, and Tsutomu Ikegami. Speed-Up Single Shot Detector on GPU with CUDA, *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, Springer International Publishing, pages 89–106, 2022.

[3] Chenyu Wang, Toshio Endo, Takahiro Hirofuchi, and Tsutomu Ikegami. Pyramid Swin Transformer: Different-size windows Swin Transformer for image classification and object detection. *In Proceedings of the 18th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications(VISIGRAPP)*, pages 583–590, SCITEPRESS, 2023. (Poster)

[4] Chenyu Wang, Toshio Endo, Takahiro Hirofuchi, and Tsutomu Ikegami. Pyramid Swin Transformer for Multi-Task: Expanding to more computer vision tasks. *Advanced Concepts for Intelligent Vision Systems(Acivs)*, 2023.