University of Padova

Department of Information Engineering

Master Thesis in ICT for Internet and Multimedia

Characterization of the Energy Consumption of the NVIDIA Edge Devices

Supervisor
Prof. Michele Rossi
University of Padova

Master Candidate
Büşra Yildiz

Student ID
2005621

Academic Year
2022-2023

Abstract

The advancements in deep neural networks have greatly impacted various fields, including computer vision, leading to their widespread usage in modern applications and systems. Despite this success, performing in-the-edge inference remains a major challenge due to the mismatch between the resource-intensive nature of deep neural networks and the limited resources available on edge devices. However, in-the-edge inferencing offers privacy benefits in user-centric domains and is crucial for scenarios with limited Internet connectivity, such as drones, robots, and autonomous vehicles. To address this, several companies have developed specialized edge devices to improve the performance of deep neural networks. This study aims to characterize several commercial edge devices on popular frameworks using the YOLO object recognition algorithm, a type of deep neural network. The impact of the framework, software stack, and optimizations on performance are analyzed, and the energy consumption of the devices is measured.

Contents

Listing of figures

Listing of tables

Listing of acronyms

AI	Artificial Intelligence
CNN	Convolutional Neural Networks
ANN	Artificial Neural Networks
ReLU	Rectified Linear Unit
R-CNN	Region-Based Convolutional Neural Networks
SVM	Support Vector Machine
ResNet	Residual Network
RPN	Region Proposal Network
YOLO	You Only Look Once
UAV	Unmanned Aerial Vehicles
IoT	Internet of Things
DNN	Deep Neural Networks
ML	Machine Learning
DL	Deep Learning
IoU	Intersection Over Union
NMS	Non Maximum Suppression
SSE	Sum Squared Error
SAT	Self Adversarial Training
DAC	Dynamic Anchor Clustering
DLS	Dynamic Layer-wise Scaling
FCN	Fully Convolutional Network
AP	Average Precision

1 Introduction

Deep learning has been a critical area of focus in the field of Artificial Intelligence (AI) over the past decade. One of the key sub-domains of deep learning is Object Detection, which is considered to be a noteworthy area in both deep learning and computer vision.

Object detection is a crucial aspect of computer vision and is accomplished through the use of neural networks. This process involves two steps, namely classification, and localization. The first step, classification, involves the algorithm identifying and categorizing individual images or sets of images and assigning labels to them. The second step, localization, involves the algorithm locating a specific object in an image and surrounding it with a bounding box to distinguish it from other objects in the image.[1].

Object Detection technology has numerous applications, including object tracking, retrieval, video surveillance, image captioning, image segmentation, object detection in retail, autonomous driving, agricultural technologies, security, medical imaging, inventory management, anomaly detection, and many others. This is due to its ability to detect objects such as humans, buildings, and cars in images and videos, making it an imperative technology in the field of computer vision.

Modern meta architectures all use CNN for object detection, let us have a look at the history of a few meta architectures briefly, and have an in-depth look at their inner working.

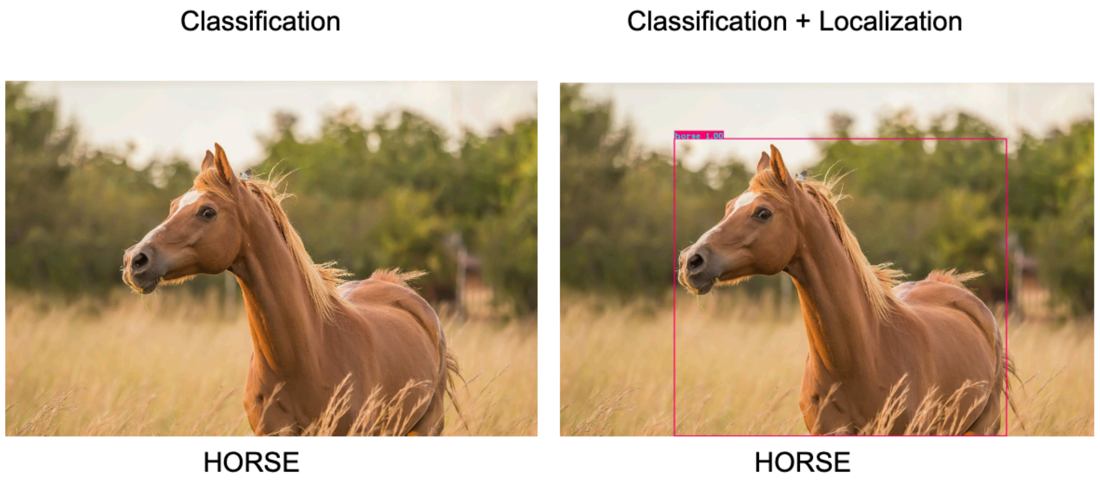


Figure 1.1: Classification and Localization

Artificial Neural Networks (ANNs) is a model that is designed to mimic the attitude of biological neural networks, such as those found in the neural system of the brain. ANNs are made up of interconnected computational nodes or neurons that work together in a distributed manner to learn from input and produce output. The typical structure of an ANN can be modeled as three layers: the input layer, hidden layers, and output layer. The input layer receives the input, which is usually in the form of a multidimensional vector, and distributes it to the hidden layers. The hidden layers then make decisions based on the previous layer and weigh how changes within themselves affect the final output. This process of learning is essential to ANNs. When multiple hidden layers are stacked on top of each other, it is commonly referred to as deep learning. The field of machine learning and artificial intelligence can be broadly divided into two categories, namely supervised learning and unsupervised learning. Supervised learning involves training a model using labeled datasets, where each training example has a set of input values and one or more designated output values. The objective is to minimize the classification error of the model by correctly predicting the output value for each training example. On the other hand, unsupervised learning is a type of learning where the training dataset is unlabeled. The performance of the model is measured by its ability to minimize or maximize an associated cost function. However, it's worth noting that most image-based pattern recognition tasks rely on supervised learning for classification purposes.[2]

A CNN, or Convolutional Neural Network, is a type of deep learning model designed to process data with a grid-like structure, such as images or audio spectrograms. The network

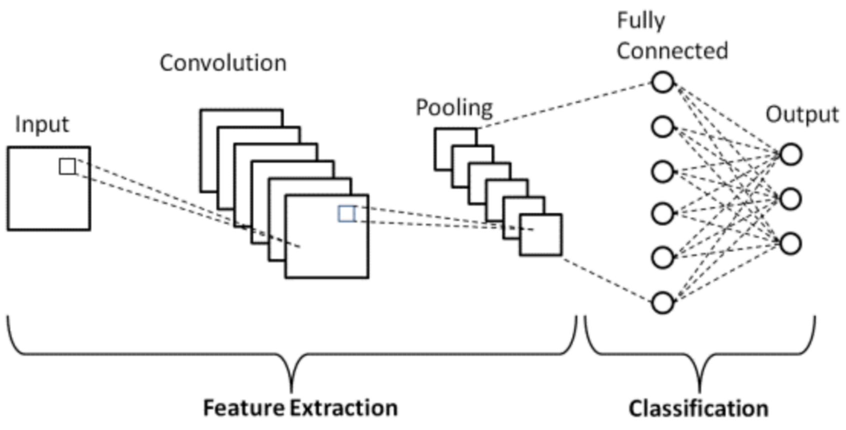


Figure 1.2: CNN Architecture

consists of three types of layers: convolution, pooling, and fully connected layers. The convolution and pooling layers perform feature extraction, while the fully connected layer maps the extracted features into the final output, such as classification.

In a CNN, a convolution layer plays a key role by applying a set of filters, also known as kernels, to the input data in a stack of mathematical operations. These kernels, which are optimizable feature extractors, are trained to extract features from the input data, which can hierarchically and progressively become more complex as one layer feeds its output into the next layer. The training process involves adjusting the weights of the filters to minimize the difference between the predicted output of the network and the true labels of the input data, using optimization algorithms like backpropagation and gradient descent.

After the convolutional layer, a non-linear activation function such as the rectified linear unit (ReLU) is applied to introduce non-linearity into the model. The output is then passed through a pooling layer, which reduces the spatial dimensionality of the output by taking the maximum or average value within a small window and discarding the rest.

Finally, the output of the pooling layer is passed to one or more fully connected layers, which perform a classification or regression task on the extracted features. In a classification task, the final layer of the network typically uses a softmax activation function to produce a probability distribution over the classes. Overall, CNNs are particularly effective for tasks such as image classification and object detection in computer vision due to their ability to learn local and spatial patterns in the input data.[3]

The R-CNN model is one of the earliest approaches to object detection that utilizes convolutional neural networks. Its objective is to take an image as input and precisely locate the primary objects within the image through bounding boxes. To accomplish this, R-CNN proposes multiple boxes in the image and determines which of them corresponds to an object. The process of generating these region proposals, known as Selective Search, involves examining the image through windows of varying sizes and grouping adjacent pixels based on texture, color, or intensity to identify objects. After the region proposals are generated, they are transformed into standard square shapes and passed through a feature extractor or image classifier, which is a CNN. The final layer of the CNN includes a Support Vector Machine (SVM) that classifies whether the region corresponds to an object and if so, identifies the specific object.[4]

R-CNN works really well but is really quite slow for a few simple reasons. Some of the drawbacks of R-CNN to build a faster object detection algorithm were solved and it was called Fast R-CNN. The approach is similar to the R-CNN algorithm. But, instead of feeding the region proposals to the CNN, we feed the input image to the CNN to generate a convolutional feature map. From the convolutional feature map, we identify the region of proposals and warp them into squares, and by using an RoI pooling layer we reshape them into a fixed size so that they can be fed into a fully connected layer. From the RoI feature vector, we use a softmax layer to predict the class of the proposed region and also the offset values for the bounding box. The reason "Fast R-CNN" is faster than R-CNN is that you don't have to feed region proposals to the convolutional neural network every time. Instead, the convolution operation is done only once per image and a feature map is generated from it.[5]

Fast R-CNN outperforms R-CNN, but its performance was surpassed by the development of an even more advanced object detection algorithm, Faster R-CNN. The problem with R-CNN and Fast R-CNN was the use of selective search to identify the region proposals, which was slow and negatively impacted the performance of the network. The solution was to create an algorithm that eliminates the selective search and allows the network to learn the region proposals on its own. [6]

This was achieved by providing the image as input to a convolutional network, which generates a convolutional feature map. Instead of relying on the selective search to identify the region proposals, a separate network was used to predict them. The predicted region proposals were then transformed using RoI pooling, which was used to classify the image within the region and predict the bounding box offset values.

ResNet is another architecture used in the field of computer vision. ResNet is a deep neural network architecture that was introduced to solve the problem of vanishing gradients in very deep neural networks. It uses residual connections to enable information to bypass a layer and flow directly to the next layer, allowing the network to learn more complex features and overcome the vanishing gradient problem. ResNets are particularly useful for image recognition tasks, where they can achieve state-of-the-art accuracy with fewer layers than traditional neural network architectures.

On the other hand, Faster R-CNN is a framework for object detection that consists of two main components: a region proposal network (RPN) that generates candidate object regions, and a detection network that classifies the proposed regions and refines their locations. Faster R-CNN is designed for the task of object detection, where the goal is to identify the presence and location of objects in an image.

Both ResNet and Faster R-CNN have their own strengths and weaknesses, and the choice between them depends on the specific task at hand. If the task is image recognition, ResNet is a better choice due to its superior performance in this area. However, if the task is object detection, Faster R-CNN is more appropriate as it is specifically designed for this task.

So far, all object detection algorithms utilize regions to determine the location of the object within the image. Instead of examining the entire image, the network focuses on parts of the image that have a high probability of containing the object. YOLO (You Only Look Once) is an object detection algorithm that differs significantly from region-based algorithms. With YOLO, a single convolutional network predicts both the bounding boxes and the class probabilities for these boxes. The image is divided into an SxS grid, and within each grid, there are m bounding boxes. The network outputs class probabilities and offsets values for each bounding box. The boxes with class probabilities above a certain threshold are selected and used to locate the object in the image. YOLO is much faster than other object detection algorithms, but it has limitations when it comes to detecting small objects due to its spatial constraints.

However, the execution platform of several of these applications is limited in terms of resources for efficient algorithm execution. These platforms, such as low-cost robots, unmanned aerial vehicles (UAVs), and Internet of Things (IoT) devices, are unable to meet the resource requirements for fast prediction of object detection algorithms, which necessitates high energy consumption, ample memory, and robust processors. The conventional method to address

Method	mAP-50	Inference Time (ms)
SSD321	45.4	61
DSSD321	46.1	85
R-FCN	51.9	85
SSD513	50.4	125
DSSD513	53.3	156
FPN FRCN	59.1	172
RetinaNet-50-500	50.9	73
RetinaNet-101-500	53.1	90
RetinaNet-101-800	57.5	198
YOLOv3-320	51.5	22
YOLOv3-416	55.3	29
YOLOv3-618	57.9	51

Table 1.1: YOLOv3 vs Other Algorithms

this issue is to offload all computation to a cloud environment. However, this is not a viable solution in certain cases due to privacy considerations, limitations in internet connectivity, or strict time constraints.

Companies have started to build device-specific frameworks for efficient Deep Neural Network(DNN) execution with several compilers and software-level optimizations. However, using only software techniques cannot guarantee the fast execution of DNNs. This is because current hardware platforms are not specifically designed for DNNs, the execution of which has unique characteristics. This inefficiency of general-purpose hardware platforms Additionally, companies have also released specialized accelerator edge devices for performing fast in-the-edge inferencing. [7]

Edge computing is an emerging computing paradigm that refers to a range of networks and devices at or near the user. Edge is about processing data closer to where it's being generated, enabling processing at greater speeds and volumes, leading to greater action-led results in real-time. [8]

AI edge computing, AI applications to run directly on field devices, processing field data, and run machine learning (ML) and deep learning (DL) algorithms.

This study presents a benchmarking between commercial edge devices (such as Nvidia's Jetson Nano, Jetson TX2, and Jetson Xavier) with the same set of assumptions among various versions of YOLO which is one of the Object Detection algorithms.

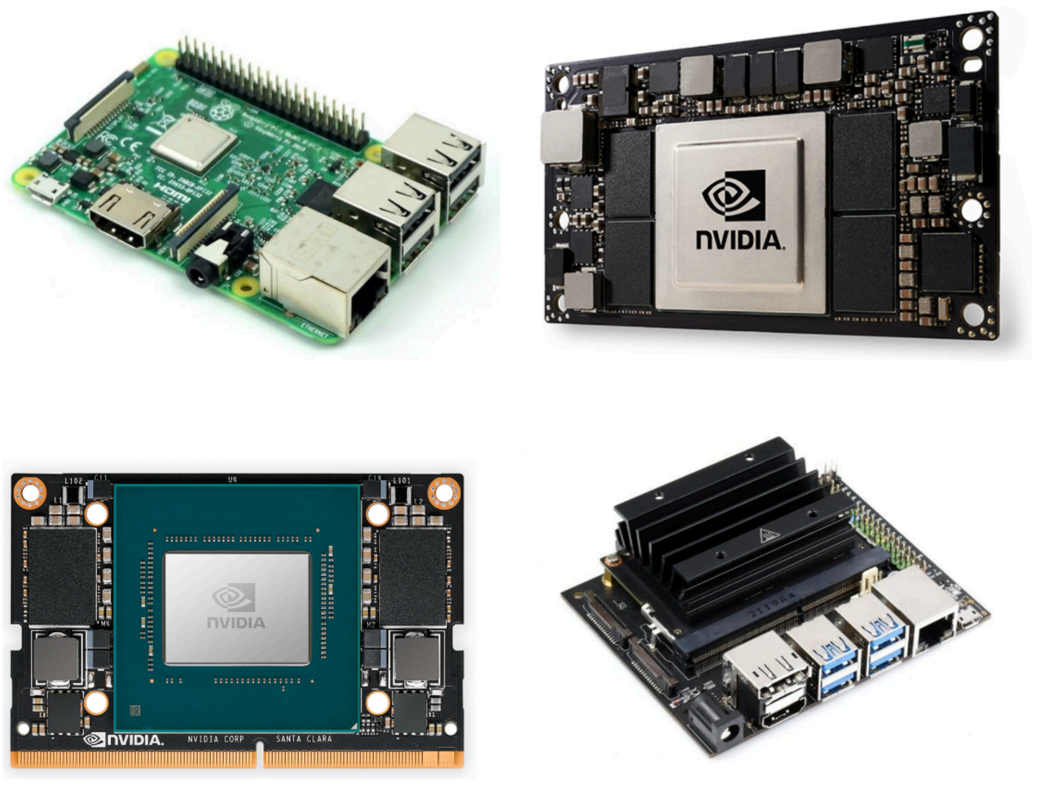


Figure 1.3: Some Edge Devices

2 Related Works

Characterizing the single board computers and making benchmarking them has been a very interesting research area lately, and various approaches have been developed. In this section, we will focus on different approaches to benchmarking several edge devices.

For instance, In this study, they compared performances of single-board computers in NVIDIA Jetson Nano, NVIDIA Jetson TX2, and Raspberry PI4 through the CNN algorithm created by using the fashion product images dataset. For this, they developed a 2D-CNN model. They trained and tested it on five different-sized datasets with NVIDIA Jetson Nano, NVIDIA Jetson TX2, and Raspberry PI 4. [9]

It was observed that as the volume of big data increased in Jetson TX, the power consumption became more stable. Further tests conducted on the impact of dataset size on the accuracy of deep learning applications revealed a positive correlation. The results indicated that the model trained on a dataset of 45,000 items using Jetson TX2 achieved the highest accuracy of 97.8%. While the study noted that the Jetson TX2 consumed more power, it also outperformed other systems in terms of accuracy, faster processing time, and larger dataset-handling capabilities.

A different study investigated the inference workflow and performance of the You Only Look Once (YOLO) network on three different single-board computers: the NVIDIA Jetson Nano, NVIDIA Jetson Xavier NX, and Raspberry Pi 4B (RPi). By comparing the inference performance of these three SBCs, the study found that the performance of the RPi + NCS2(Intel Neural Compute Stick2) is better suited for lightweight models. Therefore, the study suggests that the Jetson Nano is a cost-performance trade-off among the SBCs, as it can achieve up to 15 FPSs of detected videos when running YOLOv4-tiny.[10]

Similarly, in another study that analyzed the impact of frameworks, software stacks, and applied optimizations on the final performance, the energy consumption and temperature behavior of these edge devices were measured. In their study, they used various DNN models (ResNet-18, ResNet-50, ResNet-101, Xception, MobileNet-v2, Inception-v4, AlexNet, VGG16, VGG19, VGG-S, VGG-S, CifarNet, SSD with MobileNet-v1, YOLOv3, TinyYolo, C3D) and benchmarked on the following devices : Raspberry Pi 3B (IoT/Edge Device), Jetson TX2 and Jetson Nano (GPU-Based Edge Devices), EdgeTPU and Movidius NCS (Custom-ASIC Edge Accelerators), PYNQ-Z1 (FPGA based device), Xeon, RTX 2080, GTX Titan X, Titan Xp. As a result, a tradeoff was discovered between energy usage and inference time when comparing Movidius and Jetson Nano edge devices.[11]

3 Dataset

The COCO dataset is extensive and includes object detection, segmentation, and captioning capabilities. The highly acclaimed COCO dataset is a feature-rich resource, boasting advanced capabilities such as object segmentation, recognition in context, superpixel stuff segmentation, and a staggering 330K images, over 200K of which are labeled with 1.5 million object instances across 80 object categories and have 91 stuff categories, each accompanied by 5 descriptive captions per image, and a diverse representation of 250,000 individuals with keypoints annotated. [12]

Figure 3.1: Coco Dataset Examples

4
Methods

YOLO MODELS OVERVIEW

The YOLO algorithm improvement is still ongoing.In this section, first,we will examine the YOLO versions developed so far.

The new approach to object detection has been proposed by Redmon et al in 2016. In contrast to earlier approaches that used classifiers for detection, they approached object detection as a regression task, predicting separate bounding boxes and corresponding class probabilities in spatially separate.

For instance, the R-CNN, employs region proposal techniques to generate candidate bounding boxes in an image. A classifier is then used to classify the suggested boxes. Following classification, post-processing techniques are used to refine the bounding boxes, remove any duplicate detections, and rescore the boxes based on other objects in the scene. Because each component of this approach must be trained separately, it can be slow and challenging to optimize. In contrast, YOLO simplifies the process of object detection by framing it as a single regression problem that maps directly from image pixels to bounding box coordinates and class probabilities. This means that only one pass is needed over an image to identify the objects present and their locations.

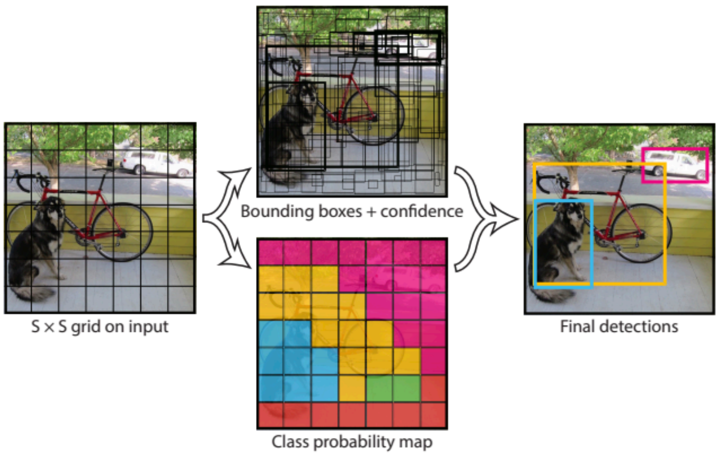


Figure 4.1

The YOLO algorithm partitions an input image into a grid of fixed size, typically denoted as S x S. Each grid cell is then responsible for predicting the presence of objects in that cell as well as the location and class of the object.

To achieve this, each grid cell predicts B bounding boxes, where B is a hyperparameter that determines the number of boxes to be predicted. These bounding boxes contain five parameters for each box (pc,bx,by,bh,bw): the x and y coordinates of the box's center with respect to the grid cell, the width and height of the box as a fraction of the image size, and a confidence score. The confidence score reflects the probability that the predicted box contains an object and measures the accuracy of the prediction.

Additionally, each grid cell predicts an "objectness" score, P(Object), which is a probability value that determines whether an object exists in the cell or not. This score helps the algorithm to filter out false positives and reduce the number of unnecessary detections.

Moreover, each grid cell also predicts the conditional probability of the object belonging to a particular class, given that an object exists in that cell, denoted as P(Class | Object). This allows the algorithm to detect and classify multiple objects of different categories present in the same image.

The center coordinates of the bounding box are expressed with respect to the grid cell in which the box lies. These parameters are bounded between 0 and 1, indicating their fractional representation of the entire image. Specifically, the values of x and y are fractions of the cell's width and height, whereas the values of w and h are fractions of the image's overall dimensions.

The confidence score is used to determine the box's accuracy and the presence of an object.

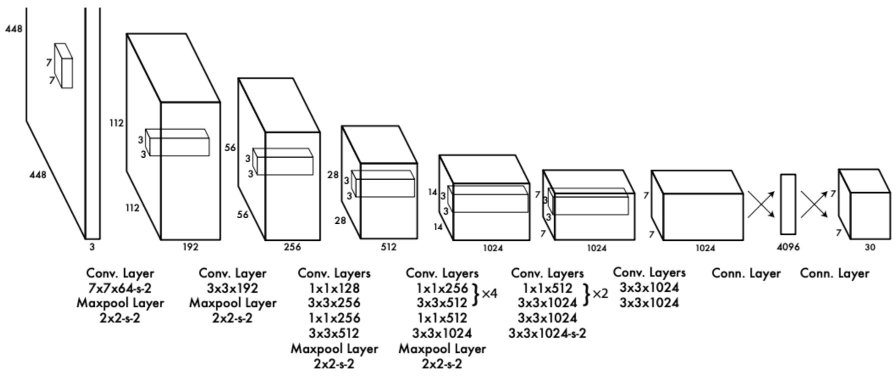


Figure 4.2: The YOLO v1 Architecture

If the box does not contain an object, the confidence score is zero. Conversely, if an object exists within the box, the confidence score equals the Intersection Over Union (IoU) between the predicted box and the ground truth.

YOLO predicts a total of B x 5 parameters for each grid cell, where B represents the number of bounding boxes predicted per cell. By accurately predicting the parameters of each bounding box, YOLO can identify and locate objects in an input image.

The algorithm predicts C class probabilities for each grid cell in an input image. These probabilities are conditional on an object being present in the grid cell. Despite having B bounding boxes per grid cell, YOLO only predicts one set of C class probabilities. This means that for each grid cell, YOLO predicts a total of C + B x 5 parameters.

The final prediction tensor for an input image is determined by the size of the grid, S, the number of bounding boxes per grid cell, B, and the number of classes, C. For instance, YOLO uses S = 7, B = 2, and C = 20 for the PASCAL VOC dataset. This results in a 7 x 7 x (20 + 5 x 2) = 7 x 7 x 30 tensor as the final YOLO prediction for PASCAL VOC.

Finally, YOLO version 1 applies Non Maximum Suppression (NMS) and thresholding to report the final predictions. This technique helps eliminate duplicate detections and refine the final output by selecting the most confident predictions.

Figure 4.2 illustrates the architecture of YOLO v1 CNN, which comprises 24 convolution layers serving as a feature extractor, followed by 2 fully connected layers responsible for object classification and bounding box regression. The network produces a 7 x 7 x 30 tensor as its final output. YOLO CNN is a straightforward CNN with a single path, and it employs 1x1 convolutions followed by 3x3 convolutions, taking inspiration from Inception version 1 CNN developed by Google. Leaky ReLU activation is used for all layers except the final layer, which utilizes a linear activation function.

In YOLO, sum-squared error (SSE) is used as the loss function for training the neural network. SSE measures the squared difference between the predicted values and the ground truth values, which includes the coordinates of the bounding box, the confidence score, and the class probabilities.

In YOLO, each grid cell is responsible for predicting a certain number of bounding boxes, depending on the value of the hyperparameter. For example, if the hyperparameter is set to 2, each grid cell will predict 2 bounding boxes. However, some grid cells may not contain any objects, and their confidence score will be set to zero. This can lead to overpowering of the gradients from the grid cells that do contain objects, which can cause training divergence and model instability.

To address this issue, YOLO increases the weight (λcoord = 5) for predictions from bounding boxes containing objects, which means that the loss from these predictions will have a greater impact on the overall loss than predictions from grid cells that do not contain objects. Conversely, YOLO reduces the weight (λnoobj = 0.5) for predictions from bounding boxes that do not contain any objects, which means that the loss from these predictions will have less impact on the overall loss.

By adjusting the weights for different types of predictions, YOLO can ensure that the model focuses more on correctly predicting the bounding boxes that contain objects, while also taking into account the predictions from grid cells that do not contain any objects. This can lead to a more stable and accurate model.

$$\lambda\text{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}1_{ij}^{obj}\left[\left(x_i-\hat{x}_i\right)^2+\left(y_i-\hat{y}_i\right)^2\right]$$

$$+\lambda\text{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}1_{ij}^{obj}\left[\left(\sqrt{w_i}-\sqrt{\hat{w}_i}\right)^2+\left(\sqrt{h_i}-\sqrt{\hat{h}_i}\right)^2\right]$$

$$+\sum_{i=0}^{S^2}\sum_{j=0}^{B}1_{ij}^{obj}\left(C_i-\hat{C}_i\right)^2$$

$$+\lambda\text{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}1_{ij}^{obj}\left(C_i-\hat{C}_i\right)^2$$

$$+\sum_{i=0}^{S^2}1_{i}^{obj}\sum_{C\varepsilon classes}\left(p_i(c)-\hat{p}_i(c)\right)^2$$

Here, we will examine the loss formula in more detail.

$$\lambda\text{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}1_{ij}^{obj}\left[\left(x_i-\hat{x}_i\right)^2+\left(y_i-\hat{y}_i\right)^2\right]$$

The formula shown above corresponds to the initial element of the YOLO loss, which measures the discrepancy in predicting the center coordinates of the bounding box. The loss function solely penalizes the error in bounding box center coordinates for the predictor that is accountable for the actual ground truth box.

$$\lambda\text{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}1_{ij}^{obj}\left[\left(\sqrt{w_i}-\sqrt{\hat{w}_i}\right)^2+\left(\sqrt{h_i}-\sqrt{\hat{h}_i}\right)^2\right]$$

The second part of the loss formula presented above calculates the error in the prediction of bounding box width and height. If the error in prediction has the same magnitude for both small and large bounding boxes, they will produce the same loss value. However, a similar magnitude of error is considered more severe for small bounding boxes than for large bounding boxes. To account for this, the square root of the error values is used to calculate the loss. Since both width and height are limited to a range of 0 to 1, taking the square root increases the difference for smaller values more than larger values. The loss function only penalizes the error in bounding box width and height if the predictor is responsible for the ground truth box.

$$\sum_{i=0}^{S^2}\sum_{j=0}^{B}1_{ij}^{obj}\left(C_i-\hat{C}_i\right)^2$$

The formula presented above is the third part that computes the error in predicting the object confidence score for bounding boxes containing an object. The loss function penalizes the error in the object confidence score only when the predictor is accountable for the ground truth box.

$$\lambda\text{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}1_{ij}^{obj}\left(_{C_i-\hat{C}_i}\right)^2$$

The fourth part of the YOLO loss formula presented above computes the error in predicting the object confidence score for bounding boxes that do not contain any objects. Similar to the previous parts of the loss formula, the model is penalized for the object confidence error only when the predictor is responsible for the ground truth box. This encourages the model to accurately distinguish between the areas in the image that contain objects and those that do not.

$$\sum_{i=0}^{S^2}1_{i}^{obj}\sum_{C\varepsilon classes}\left(_{p_i(c)}-\hat{p}_i(c)\right)^2$$

The final segment of the YOLO loss formula presented above calculates the error in predicting the class probabilities for grid cells that contain an object. In line with the previous parts of the loss formula, the model is penalized for the error in class probabilities only when an object is present in the corresponding grid cell. This helps the model to improve its accuracy in identifying the object class present in the given image.

YOLOv1 has certain limitations, such as struggling to detect small objects that appear in clusters or have unconventional aspect ratios. Additionally, it tends to have more errors in localizing objects when compared to Fast R-CNN.

In YOLO v2, the architecture includes batch normalization applied to convolutional layers. This reduces the shift in the unit value within hidden layers, resulting in improved neural network stability. The addition of batch normalization to convolutional layers has led to a 2% improvement in MAP (mean average precision) and also contributed to reducing overfitting, resulting in improved model regularization.

In YOLOv2, Input size was increased from 224*224 to 448*448. Thus, this increase in Input size resulted in up to 4% improvement in MAP.

An important update that can be observed in YOLOv2 is the incorporation of anchor boxes, which enable classification and prediction to be performed within a unified framework. The anchor boxes are utilized to forecast the bounding boxes and are customized for a specific dataset using k-means clustering.

One of the primary challenges faced by YOLOv1 was detecting smaller objects in images. YOLOv2 overcomes this issue by dividing the image into smaller 13x13 grid cells, which is a more refined grid compared to the previous version. By doing so, YOLOv2 can effectively localize and identify both smaller and larger objects in the image.

Also, In YOLOv1, a limitation was encountered while detecting objects of varying sizes. If YOLO was trained with small images of a specific object, it struggled to identify the same object in a larger image. However, this issue has largely been addressed in YOLOv2, which is trained on a range of random images with dimensions ranging from 320x320 to 608x608. As a result, the network can learn and accurately predict objects of different sizes in varying input dimensions.

The Darknet 19 architecture, which comprises 19 convolutional layers and 5 max-pooling layers, is employed in YOLO v2 to classify objects using a softmax layer. Darknet is a neural network framework that is programmed in CUDA and is renowned for its exceptional speed in object detection, making it ideal for real-time predictions.

Significant advancements have been made in some categories, leading to a marked improvement in YOLOv2's ability to detect smaller objects with greater accuracy.

The incremental improvements made in YOLOv3 involve the use of logistic classifiers for multi-label classification, as well as the incorporation of logistic regression to predict the objectiveness score for each bounding box.

In contrast to YOLOv2, which employed a softmax layer for classification, YOLOv3 uses logistic classifiers for each class, enabling the model to label an object with multiple classes. Furthermore, the use of logistic regression in YOLOv3 allows for a more precise prediction of the objectiveness score associated with each bounding box.

To illustrate the difference between softmax and logistic classifiers, consider a network trained to recognize both cats and kittens. Using softmax, the network would provide probabilities for both classes, such as 0.5 for cats and 0.58 for kittens. However, with logistic classifiers, the network would provide independent probabilities for each class. For example, if the network was trained to recognize cats and kittens, it might provide a probability of 0.85 for cats and 0.8 for kittens, thereby allowing the object in the image to be labeled as both a cat and a kitten.

In YOLO v3, predictions are generated in a similar fashion to the Feature Pyramid Networks (FPN), whereby three predictions are made for every location in the input image, and features are extracted from each prediction. This approach enables YOLOv3 to have better scalability across multiple object sizes.

As described in the research paper by [13], each prediction in YOLO v3 consists of a boundary box, objectness, and 80 class scores. By upsampling from previous layers, YOLOv3 is able to obtain full semantic information and finer-grained details from an earlier feature map, thus improving the quality of the output. The addition of a few more convolutional layers to the process further enhances the accuracy of the model.

In contrast to its predecessor YOLO v2, which employed Darknet-19 as a feature extractor, YOLO v3 utilizes Darknet-53 as its feature extractor, boasting a remarkable 53 convolutional layers. Compared to YOLO v2, this feature extractor is significantly deeper. Darknet-53 is predominantly composed of 3x3 and 1x1 filters and includes shortcut connections to facilitate information flow across the network.

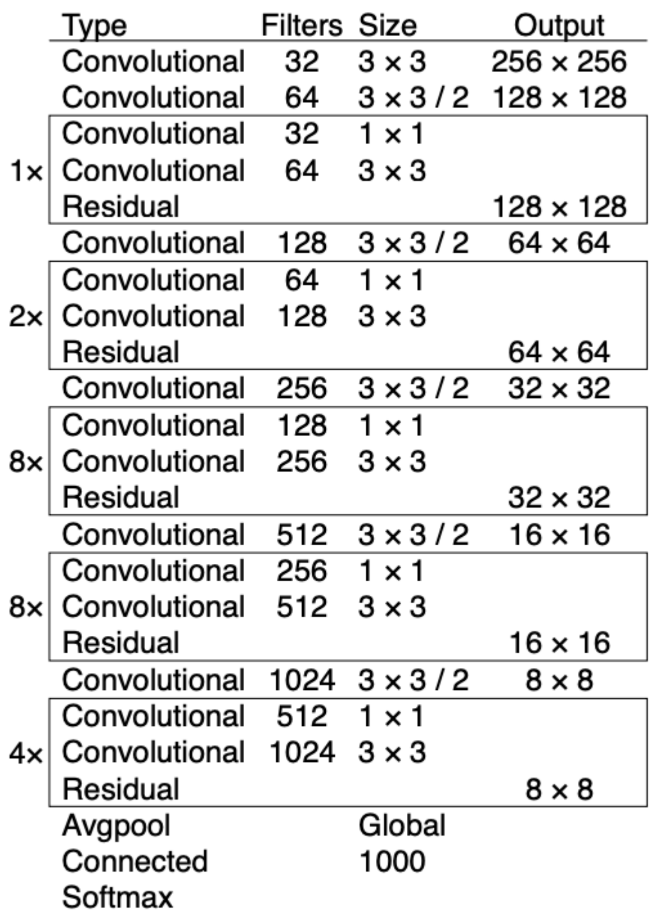
While both YOLOv3 and YOLOv4 versions use similar principles and techniques, there are some significant differences between them. One of the most notable differences between YOLOv3 and YOLOv4 is their architecture. YOLOv4 features a more complex backbone network, including Darknet53, which helps improve the overall accuracy of the model. YOLO v4 also uses a modified anchor box clustering technique, which allows the model to detect objects at different scales more accurately.

Another significant difference between the two versions is the way they handle data augmentation. YOLOv4 uses a more extensive data augmentation pipeline, which includes mosaic data augmentation, cutout data augmentation, and more. This allows the model to learn more diverse and robust features, leading to improved performance.

Also, YOLO v4 has made several improvements to its training process. For example, it uses a more advanced optimizer (the Mish activation function) and a more sophisticated learning rate scheduler. These improvements help the model train faster and more effectively, resulting in better object detection accuracy.

Overall, while YOLO v3 and YOLO v4 share many similarities, YOLO v4 represents a significant improvement over its predecessor in terms of accuracy, speed, and robustness.

One of the most significant differences between YOLO v4 and v5 is the architecture. YOLO v5 introduces a new architecture that is significantly smaller and faster than its predecessor, with fewer layers and more efficient feature extraction methods. This allows YOLO v5 to run faster on smaller devices, making it more suitable for real-time object detection applications.



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

Figure 4.3: Darknet-53

Another major difference between the two models is the training data. YOLO v5 is trained on a larger and more diverse dataset, which includes a wider variety of object classes and image resolutions. This leads to better generalization and performance on real-world applications.

YOLO v5 introduces a new data augmentation technique called CutMix, which improves the model's ability to handle occlusion and background clutter. CutMix randomly crops and pastes patches from different images together, creating new training examples that can help the model learn to detect objects more effectively.

Furthermore, YOLO v5 introduces a new approach to model scaling, called the P6/P7 feature pyramid network. This technique enables the model to detect smaller objects and improve performance on high-resolution images.

The key difference between YOLO v6 and v5 is that YOLO v6 introduces several novel techniques to improve object detection accuracy and speed.

One of the most significant improvements in YOLO v6 is the use of a new data augmentation technique called Self-Adversarial Training (SAT). SAT creates a more diverse training dataset by perturbing the image and label data in a way that can increase model robustness and reduce overfitting.

Another major improvement in YOLO v6 is the use of a novel backbone network called CSPResNeXt, which is designed to improve feature extraction efficiency and reduce model complexity. Additionally, YOLO v6 uses a more advanced anchor box clustering method, called the Dynamic Anchor Clustering (DAC) technique, to improve object detection accuracy across different scales.

YOLO v6 introduces a new approach to model compression, called Dynamic Layer-wise Scaling (DLS). DLS enables the model to dynamically adjust the scaling of different layers based on the complexity of the input, allowing for more efficient and accurate inference on low-power devices.

One of the most significant improvements in YOLO v7 is the use of a new architecture, which is said to be more efficient and accurate than previous versions. The architecture reportedly includes a new feature pyramid network, which enables the model to detect objects at different scales and resolutions more accurately.

Another improvement in YOLO v7 is the use of a new loss function, which is designed to improve model convergence and reduce the impact of noisy labels. The loss function is said to be more robust to label noise and can improve the model's ability to generalize to new data.

Also, YOLO v7 reportedly introduces a new data augmentation technique called Elastic Deformation, which can generate more diverse training data by applying random deformations to the input images.

EDGE DEVICES OVERVIEW

Unlike desktop or personal computers, single-board computer systems are a system with high performance in their simple architecture. Although single-board computers can't perform many functions that a personal computer can perform, the difference in design and construction reveals its intended use. There are many single-board computer systems that offer possibilities to develop both hardware and software and include CPU / GPU. Jetson Nano / TX2, NVIDIA Jetson Xavier NX, Raspberry Pi, BananaPi, ODYSSEY, BeagleBoard, and Asus Tinker Board are some of them. Single-board computers are widely used in many industries, including but not limited to ATMs, medical diagnostics, precision agriculture, smart home systems, and robotics. Their versatility and flexibility make them suitable for a wide range of applications, and they offer opportunities for innovation and customization in hardware and software development.[14]

In this study, we performed our experiments using NVIDIA Jetson TX2 and NVIDIA Jetson Xavier NX.

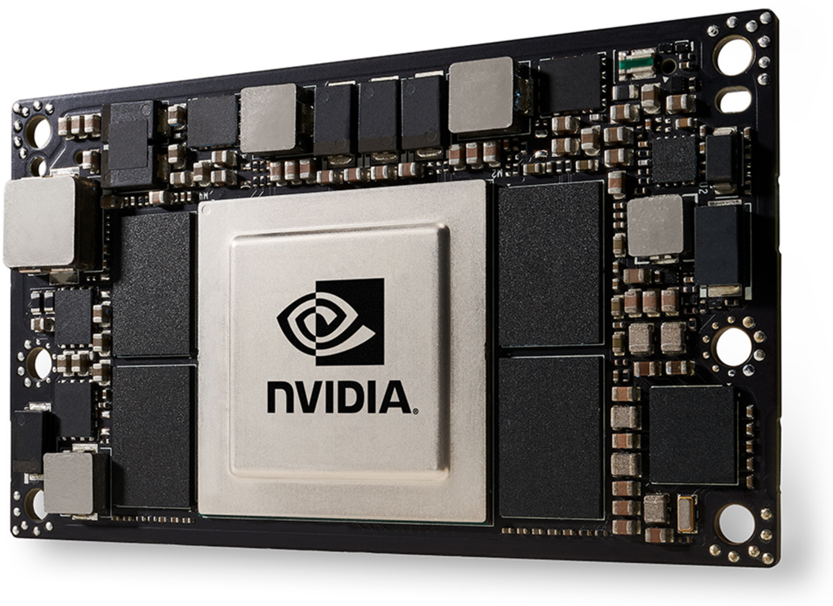


Figure 4.4: Nvidia Jetson TX2

The NVIDIA Jetson TX2 is a robust single-board computer created specifically for embedded AI applications. It is equipped with an NVIDIA Pascal GPU and a 64-bit ARM Cortex-A57 CPU with 8 cores, which enables it to provide high-performance computing for deep learning and computer vision tasks. The Jetson TX2 has 8 GB of memory and can support multiple camera inputs, making it an excellent choice for creating autonomous machines such as drones, robots, and other intelligent devices. Furthermore, its small size, low power consumption, and advanced thermal management system make it ideal for various industrial and commercial applications. In addition, the Jetson TX2 is compatible with well-known machine learning frameworks such as TensorFlow, PyTorch, and Caffe, simplifying the development and deployment of AI applications.

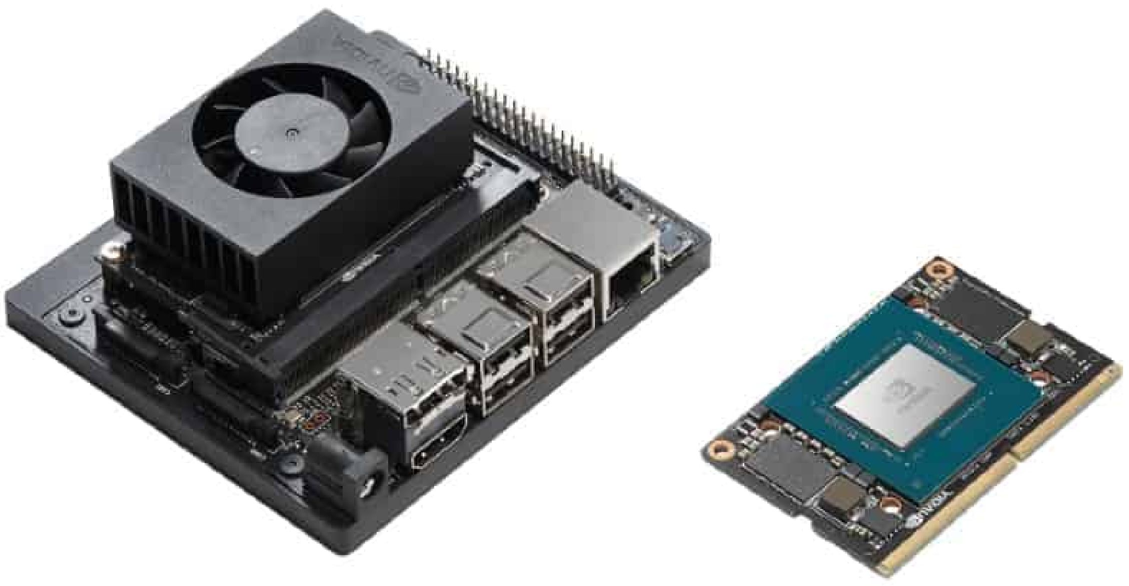


Figure 4.5: Nvidia Jetson Xavier NX

	NVIDIA Jetson Nano	NVIDIA Jetson TX2	NVIDIA Jetson Xavier NX
AI Performance	472 GFLOPs	1.33 TFLOPs	21 TOPs
GPU	128-core NVIDIA Maxwell™ GPU	256-core NVIDIA Pascal GPU	384-core NVIDIA Volta™ GPU with 48 Tensor Cores
CPU	Quad-Core Arm® Cortex®-A57 MPCore processor	Dual-Core NVIDIA Denver 2 64-Bit CPU and Quad-Core Arm® Cortex®-A57 MPCore processor	6-core NVIDIA Carmel Arm®v8.2 64-bit CPU 6MB L2 + 4MB L3
Memory	4 GB 64-bit LPDDR4 25.6GB/s	8 GB 128-bit LPDDR4 59.7GB/s	16 GB 128-bit LPDDR4x 59.7GB/s
Storage	16GB eMMC 5.1	32 GB eMMC 5.1	16 GB eMMC 5.1
Power	5W - 10W	7.5W - 15W	10W - 20W
Price	280€	635€	1900€

Table 4.1: Hardware specifications used in the benchmarks

The NVIDIA Jetson Xavier NX is a powerful and energy-efficient AI computing platform designed for edge devices and embedded systems. It features a powerful NVIDIA Volta GPU and a 6-core NVIDIA Carmel ARM v8.2 64-bit CPU, delivering up to 21 TOPS of performance at only 15W power consumption. The Jetson Xavier NX is ideal for developing and deploying AI applications in various industries, including manufacturing, healthcare, and transportation. With support for multiple cameras, sensors, and peripherals, it can perform complex deep learning and computer vision tasks in real-time. The Jetson Xavier NX is compatible with popular machine learning frameworks such as TensorFlow, PyTorch, and MXNet, and it also features support for NVIDIA's CUDA-X AI libraries, making it easy to develop and deploy AI applications. Its compact size, low power consumption, and advanced thermal management system make it ideal for use in autonomous machines such as drones and robots, as well as in intelligent surveillance systems and smart city applications.

IMPLEMENTATION

We used feature extractors called Darknet-53. Darknet-53 is a 53-layer convolutional neural network that is designed to extract features from images with high accuracy and speed. Each layer followed by bulk normalization and Leaky ReLU activation. YOLO uses only convolutional layers, which makes it a fully convolutional network (FCN), so it can avoid using pooling layers. Instead, we used a two-step convolutional layer to downsample the feature maps. This helps prevent loss of low-level features due to pooling layers.

The mesh downsamples the image with a factor called the Network pitch.The term "network pitch" refers to the scale of the grid cells used in the output feature map.

In YOLO, the input image is divided into a grid of cells, and each cell is responsible for detecting objects that fall within it. The output of the YOLO network is a feature map that corresponds to the grid cells in the input image. Each grid cell in the feature map contains a set of predicted bounding boxes and associated objectness scores.

The "network pitch" determines the size of the grid cells in the feature map. A larger pitch results in fewer cells in the feature map, which can lead to faster processing times and reduced memory requirements. However, a larger pitch may also result in reduced object detection accuracy, especially for smaller objects.

Typically, each layer in a Network reduces the output size of an image by a factor equivalent to its step. For instance, a 416 x 416 input image would produce a 13 x 13 output if the stride is 32. Similarly, a stride of 16 would result in a 26 x 26 output for the same input image, and a stride of 8 would generate a 52 x 52 output. The Network selects the cell on the input image containing the center of the ground truth box of an object to be responsible for predicting the object. A ground truth box is a bounding box annotation that specifies the precise location and dimensions of an object in an image or video frame.

During the training phase of YOLO, the algorithm is fed a dataset of images along with their corresponding ground truth box annotations. We used the COCO dataset for this.The YOLO algorithm then learns to detect objects by predicting bounding boxes around objects in new, unseen images.

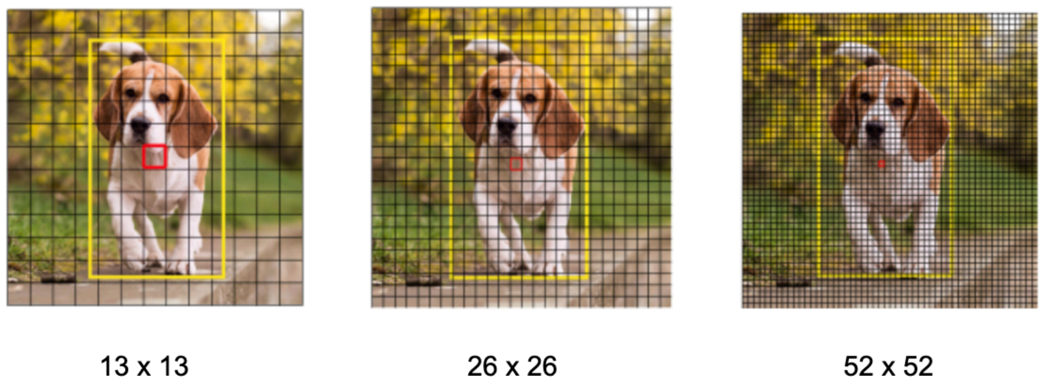


Figure 4.6: Prediction feature maps in the same input size and at different scales

The ground truth boxes in YOLO are typically represented as a set of four coordinates: the x and y coordinates of the top-left corner of the bounding box, as well as the width and height of the box. These coordinates are used to define the region of the image that contains the object of interest.

Once the YOLO algorithm has predicted bounding boxes around objects in an image, the accuracy of the algorithm can be evaluated by comparing its predictions to the ground truth boxes. This comparison helps to assess the algorithm's ability to correctly identify and locate objects in images.[15]

In figure 4.4, the cell marked red is which contains the center of the ground truth box marked yellow.

This cell can predict more than one bounding boxes. We used the anchor concept to determine which to assign to the dog image's ground truth.

Each anchor box is associated with a particular class of object and represents a prior expectation of the size and shape of objects in the image. By using multiple anchor boxes with different sizes and aspect ratios, YOLO can handle objects of different shapes and sizes.

During training, YOLO adjusts the predicted bounding boxes based on their overlap with the anchor boxes. The algorithm learns to adjust the anchor boxes to better fit the objects in the image, and to predict new bounding boxes that more accurately localize the objects.

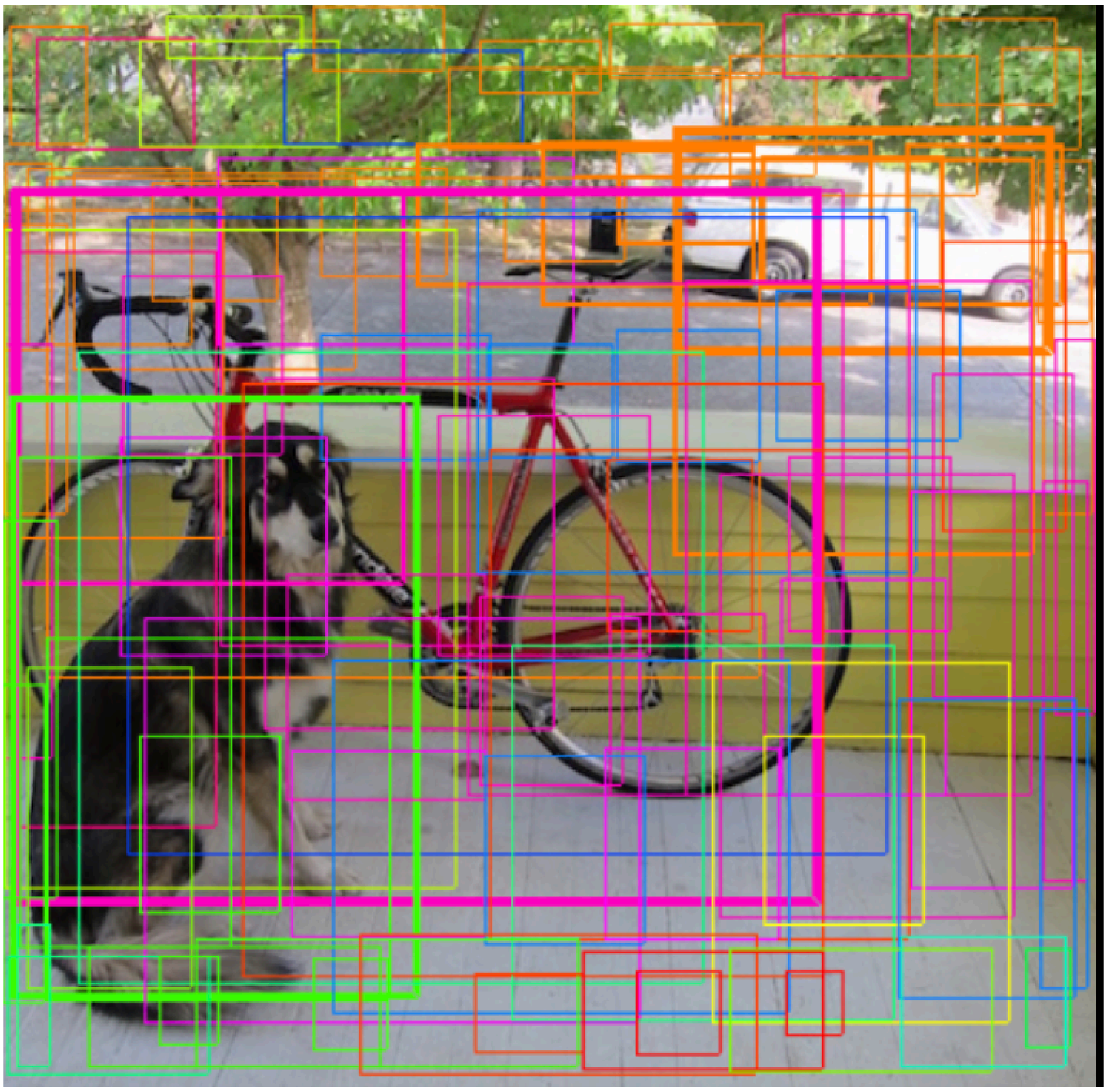


Figure 4.7: Bounding boxes

In the figure 4.7, the model has assigned high probabilities to several boxes, but there are still too many boxes to take into account. Therefore, we need to reduce the number of detected objects by filtering the algorithm's output. To accomplish this, we utilize a technique called non-maximum suppression.

In the context of the algorithm, we used log-space transformation to predict the coordinates of the bounding boxes that surround the objects in an image.

Specifically, YOLO predicts the bounding box coordinates in the form of four values: x, y, width, and height. These values are initially predicted in the linear space, but then they are transformed into the log-space using the natural logarithm function.

The reason for this transformation is to make the predictions more stable and to prevent the predictions from exploding or becoming too small. By taking the logarithm of the bounding box coordinates, the range of values that YOLO needs to predict is reduced, which makes the training process more consistent and helps to prevent overfitting.

Once the predictions have been made in the log-space, YOLO can then use the inverse logarithm function to transform them back to the original linear space. This allows the algorithm to output the final predicted bounding boxes with their actual coordinates in the image.[15]

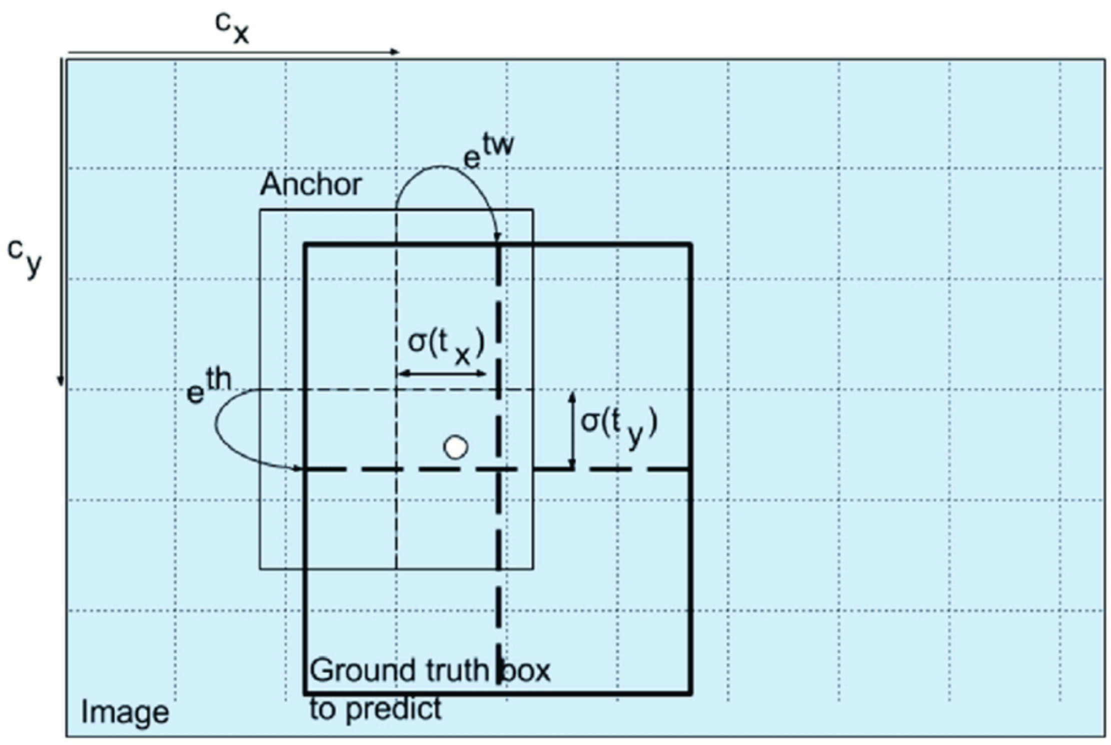


Figure 4.8: Dimensions of the Bounding box

There are two important scores that are used to predict and classify objects in an image: the objectness score and the class confidence.

The objectness score represents the probability that an object is present inside a bounding box. It should be nearly 1 for the red and the neighboring grids, whereas almost 0 for the grid at the corners. The objectness score is also passed through a sigmoid, which can be interpreted as a probability.

Confidences represent the probabilities of the detected object belonging to a particular class, such as a dog, cat, person, car, bicycle, etc. In older YOLO versions, the softmax activation function was used to calculate the class scores, but in YOLO, authors decided to use sigmoid instead. The reason for this is that soft-maxing class scores assume that the classes are mutually exclusive. To put it another way, the softmax function implies that an object must be exclusively categorized into a single class and cannot be categorized into more than one class at the same time. This is true for the COCO database, which is what YOLO is initially trained on.[15]

The class confidence scores are calculated based on the features of the object within the bounding box, such as its shape, texture, and color, and are used to classify the object into one of the predefined categories. The final output of YOLO is a set of bounding boxes, each with its associated objectness score and class confidence scores, which are used to predict and classify objects in the image. The results can be further refined using non-maximum suppression to eliminate duplicate or overlapping detections.

After filtering boxes based on their objectness score, boxes with scores below a certain threshold are typically discarded. In our experiments, we tested two different IoU thresholds: 0.25 and 0.50. To address the problem of multiple detections of the same object, Non-maximum Suppression (NMS) was used. For instance, when multiple bounding boxes in the same red grid cell or adjacent cells detect the same object, NMS is applied to eliminate the redundant detections.

To be more specific, we followed these steps:

1)Eliminated boxes with a low score.

2)Applied Non-maximum Suppression to select only one box when multiple boxes overlap and detect the same object.[15]

The Non-maximum Suppression algorithm utilizes a crucial function known as Intersection over Union. The IoU (Intersection over Union) threshold is a value used to determine how much overlap is required between two bounding boxes for them to be considered as detecting the same object. It is calculated by dividing the area of intersection between the two boxes by the area of their union.

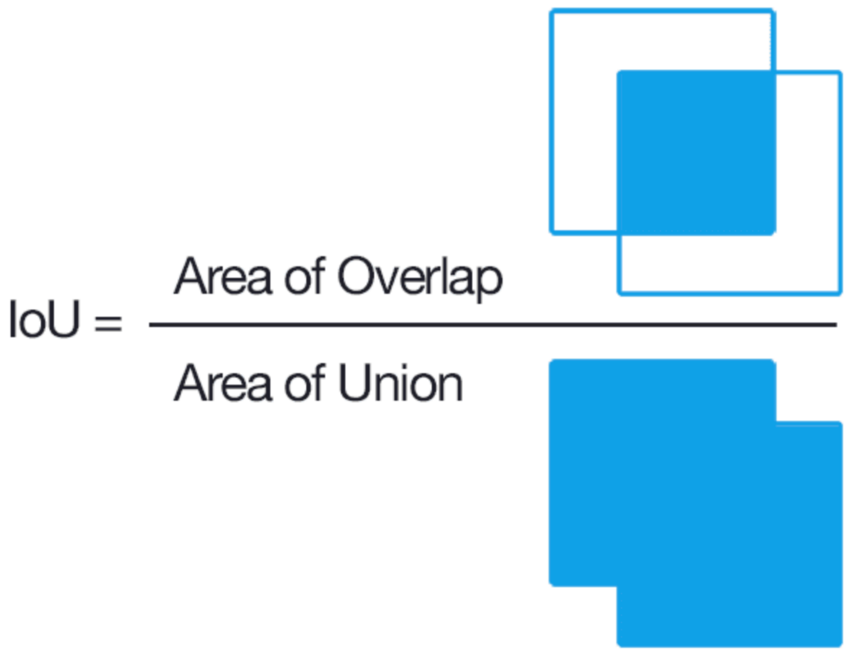


Figure 4.9: Intersection Over Union

When implementing Intersection over Union (IoU), we must first define a box using its two corners, namely the upper left and lower right points, represented by (x1, y1, x2, y2), rather than the midpoint and height/width. To calculate the area of a rectangle, we need to multiply its height (y2 - y1) by its width (x2 - x1). In addition to this, we must find the coordinates (xi1, yi1, xi2, yi2) of the intersection of two boxes. This involves identifying the maximum of the x1 and y1 coordinates of the two boxes, as well as the minimum of the x2 and y2 coordinates of the two boxes, which are then assigned as xi1, yi1, xi2, and yi2 respectively. Through the use of these approaches, we can accurately compute the IoU between two bounding boxes, which is a critical component in many computer vision tasks such as object detection and tracking.

To compute Precision and Recall, suppose the IoU threshold is set at 0.5. In that case, if the IoU value for a given prediction exceeds this threshold, typically denoted by 0.7, we classify it as a True Positive (TF). Conversely, if the IoU value falls below the threshold, say 0.3, we

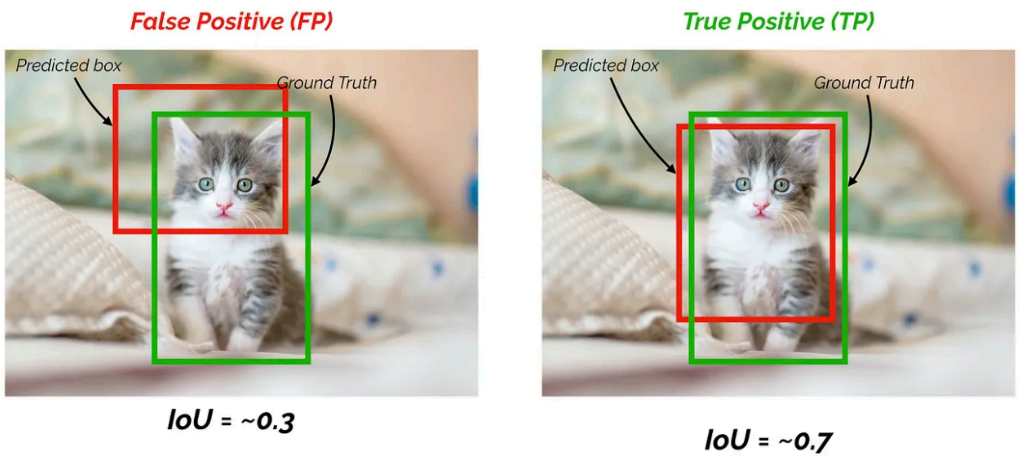


Figure 4.10: True Positive and False Positive

categorize it as a False Positive (FP). These calculations based on IoU values help evaluate the accuracy of the predictions.

An essential concept to comprehend is Recall, which assesses how effectively you locate all the positives.

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives}$$

Figure 4.11: Recall calculation

Considering Figure 4.7, the output will be as follows after applying Non-Max Suppression.

Average Precision (AP) is a performance metric commonly used in information retrieval and machine learning. It measures the area under the Precision-Recall curve, which is a graph that shows the relationship between the precision (the fraction of relevant instances among the retrieved instances) and the recall (the fraction of relevant instances that are successfully retrieved) at different classification thresholds. In other words, AP is a single number that summarizes the overall quality of a binary classification model's ability to rank and retrieve relevant instances. A higher AP indicates better performance, with a perfect AP of 1 indicating that all relevant instances were retrieved in the correct order.

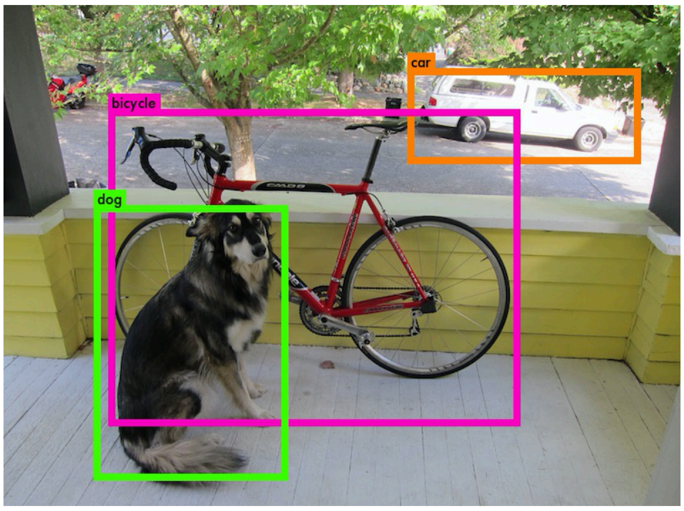


Figure 4.12: After Applying Non-Max Suppression

The mAP stands for mean Average Precision. It is a commonly used performance metric in the field of object detection. mAP is the mean of the APs calculated for each class in a multi-class problem, or the mean of APs calculated for each detection threshold in a single-class problem.

To compute the mAP for a multi-class problem, first, the AP is calculated for each class separately, and then the mean of those APs is computed to give the mAP. Similarly, in a single-class problem, the AP is calculated for each detection threshold, and then the mean of those APs is taken to give the mAP.

In essence, the mAP provides an overall assessment of how well an object detection model performs across all classes or detection thresholds, and it is a useful measure for comparing different models or configurations.

5 Results

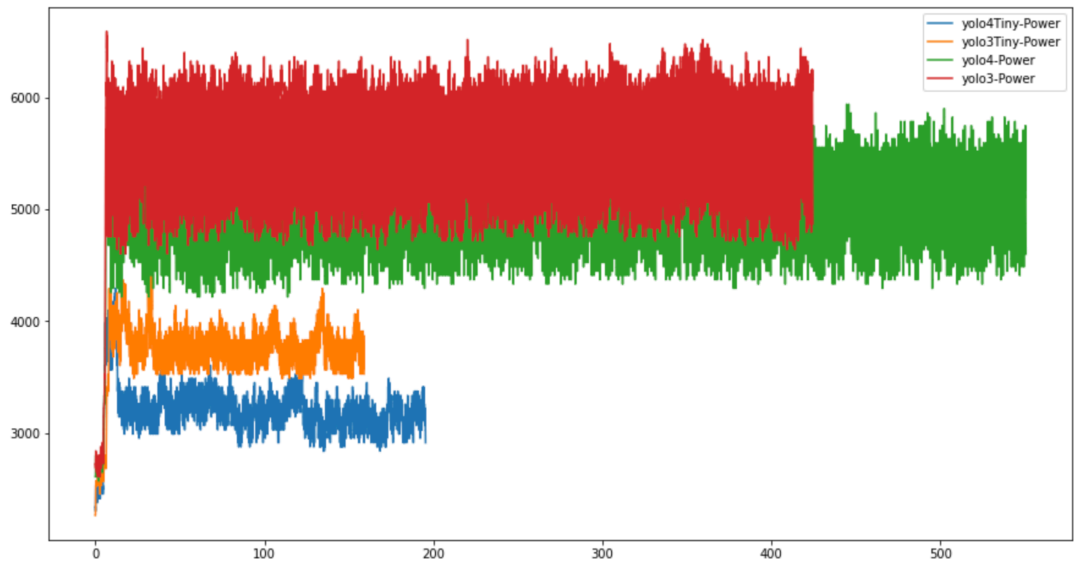


Figure 5.1: Board Power comparison of the YOLO models on NVIDIA Jetson TX2

The graph in Figure 5.1 represents the comparison of the average board power consumption for four different versions of the YOLO algorithm on NVIDIA Jetson TX2: YOLOv3, YOLOv3-tiny, YOLOv4, and YOLOv4-tiny.

The graph shows that the average board power consumption of YOLOv3-tiny and YOLOv4-tiny are very similar, and they both consume less power than YOLOv3 and YOLOv4. YOLOv3 and YOLOv4, on the other hand, have a slightly higher average power consumption.

Overall, the graph suggests that YOLOv3-tiny and YOLOv4-tiny may be more power-efficient than the YOLOv3 and YOLOv4.

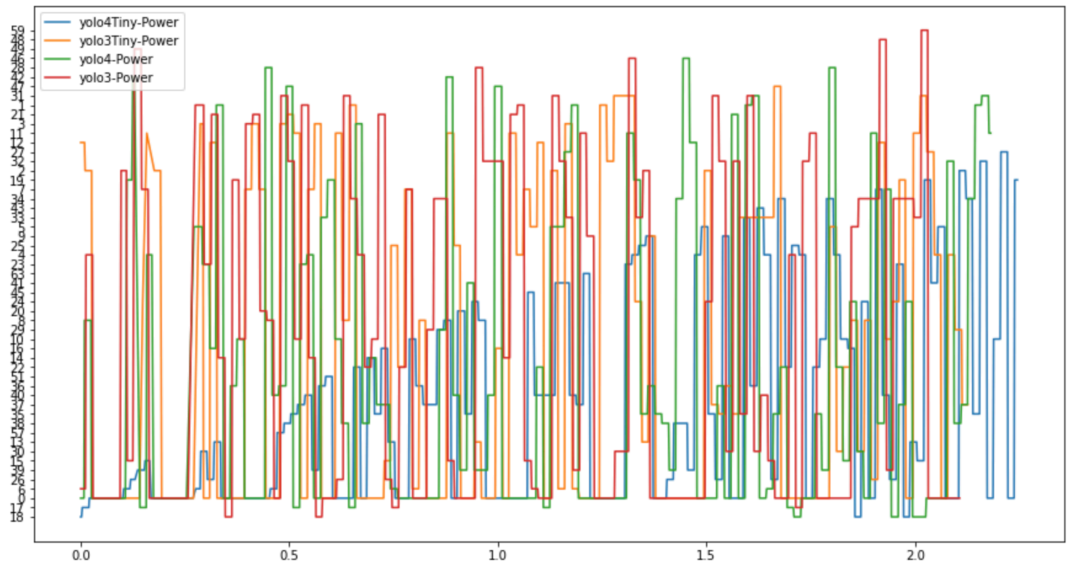


Figure 5.2: GPU Usage comparison of the YOLO models on NVIDIA Jetson TX2

The graph in Figure 5.2 shows a comparison of the GPU usage of four versions of the YOLO: YOLOv3, YOLOv3-tiny, YOLOv4, and YOLOv4-tiny.

The x-axis of the graph represents the number of input images processed per second (frames per second or fps) and the y-axis represents the GPU memory usage in gigabytes (GB).

YOLOv3-tiny has the lowest GPU memory usage among the four models but also has the lowest fps rate. This model might be a good choice if you have limited GPU memory resources and don't need to process images quickly.

YOLOv3 has higher GPU memory usage compared to YOLOv3-tiny but also has a higher fps rate. This model might be a good choice if you need a higher fps rate and have enough GPU memory to support it.

YOLOv4 and YOLOv4-tiny have the highest fps rates among the four models, but also have the highest GPU memory usage. These models might be a good choice if you need to process images quickly and have a powerful GPU with enough memory to support them.

In summary, the choice of which YOLO model to use depends on your specific needs, including the required fps rate and available GPU memory resources.

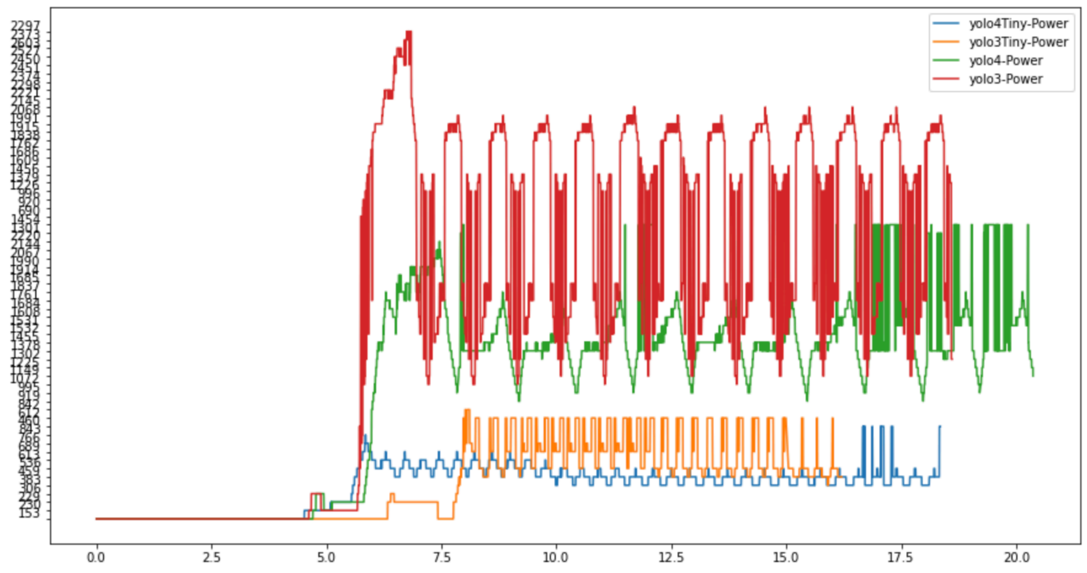


Figure 5.3: GPU Power comparison of the YOLO models on NVIDIA Jetson TX2

The graph in Figure 5.3 shows the GPU power comparison of four different versions of the YOLO object detection model: YOLOv3, YOLOv3-tiny, YOLOv4, and YOLOv4Tiny.

The x-axis of the graph represents the number of frames per second (FPS) that each model can process, while the y-axis represents the GPU power consumption measured in watts. The graph shows the relationship between FPS and power consumption for each model.

From the graph, we can see that YOLOv4 and YOLOv4Tiny have higher FPS rates compared to YOLOv3 and YOLOv3-tiny. Additionally, YOLOv4 and YOLOv4Tiny require less GPU power consumption compared to YOLOv3 and YOLOv3-tiny to achieve their respective FPS rates.

In other words, YOLOv4 and YOLOv4Tiny offer better performance and efficiency compared to YOLOv3 and YOLOv3-tiny. However, it's worth noting that the exact FPS and power consumption values may vary depending on the specific hardware used for testing.For this specific experiment we used NVIDIA Jetson TX2.

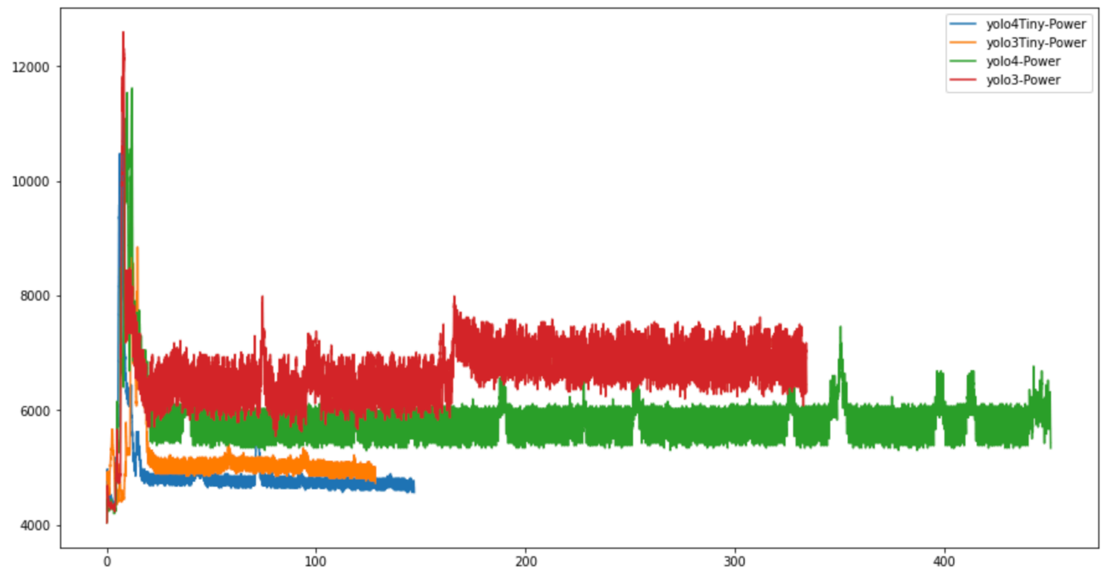


Figure 5.4: Board Power comparison of the YOLO models on NVIDIA Jetson Xavier

On the board power comparison plot, there are four versions of YOLO that were tested: YOLOv3, YOLOv3 Tiny, YOLOv4, and YOLOv4 Tiny. The Jetson Xavier was used as the hardware for the testing. The plot shows the power consumption of each version of YOLO over time during the testing process.

According to Figure 5.4, it is seen that YOLOv3 Tiny consumes the least power, followed by YOLOv4 Tiny, YOLOv3, and YOLOv4, respectively. It's also worth noting that the power consumption of each version of YOLO has changed over time with occasional increases in power usage. However, we have found that overall Tiny versions tend to consume less power. This information can be useful for individuals or organizations looking to optimize their use of YOLO while also minimizing power consumption.

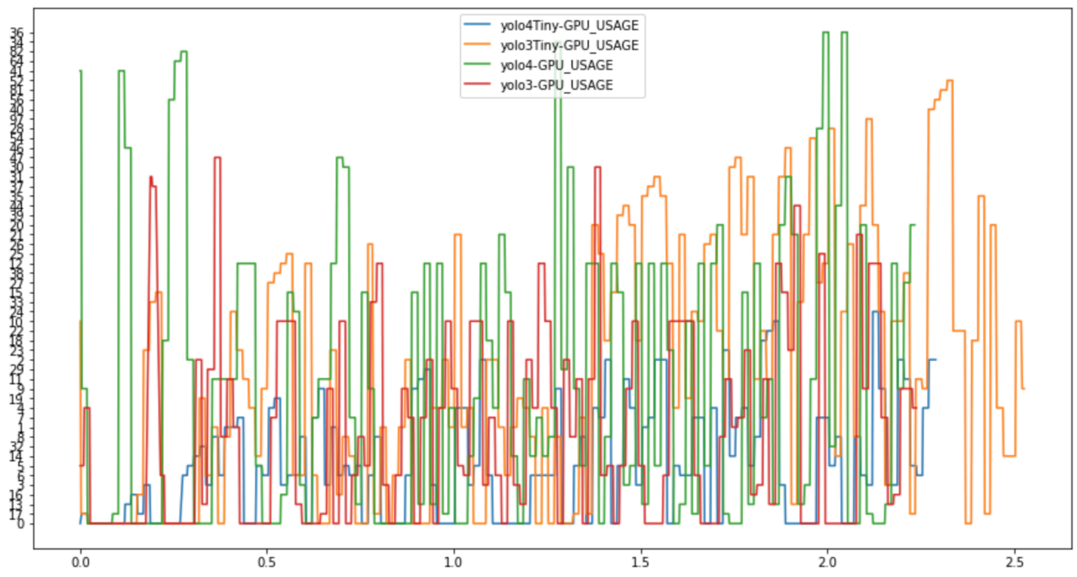


Figure 5.5: GPU Usage comparison of the YOLO models on NVIDIA Jetson Xavier

The graph in figure 5.5 shows the Jetson Xavier GPU usage comparison for four different versions of the YOLO: YOLOv3, YOLOv3-tiny, YOLOv4, and YOLOv4-tiny.

The y-axis of the graph represents the GPU memory usage in MB, and the x-axis represents the number of images processed per second. The graph shows that the YOLOv4 model uses the most GPU memory, followed by YOLOv3, YOLOv3-tiny, and YOLOv4-tiny, in that order.

In terms of processing speed, YOLOv3-tiny is the fastest among the four models, followed by YOLOv4-tiny, YOLOv3, and YOLOv4.

Overall, the graph suggests that YOLOv3-tiny may be the most efficient option for object detection tasks that require real-time processing and have limited GPU resources. However, if higher accuracy is required, YOLOv4 or YOLOv3 would be better options, but they would require more GPU memory and may have lower processing speeds.

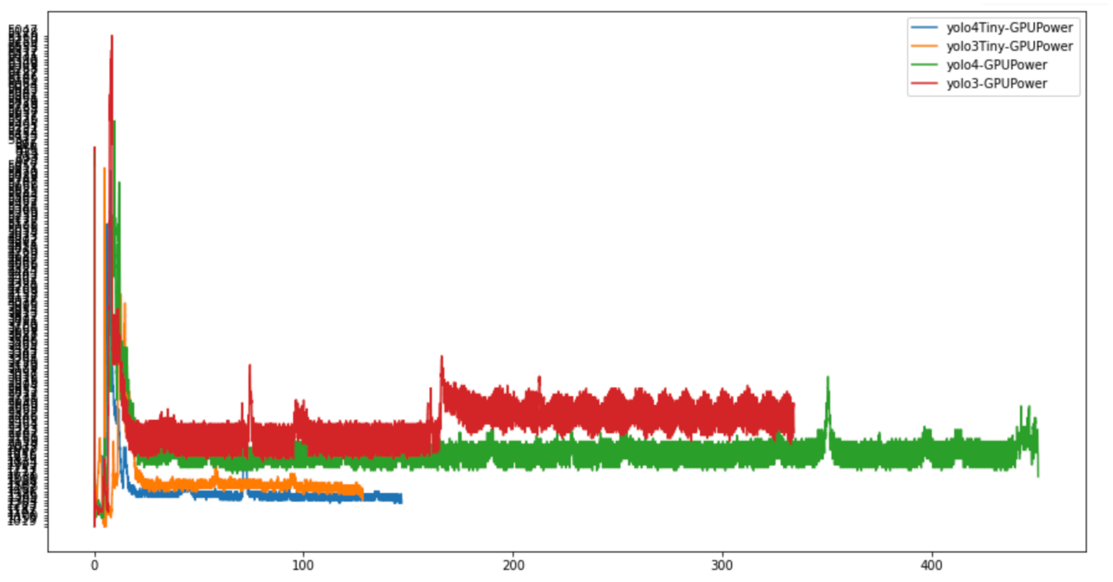


Figure 5.6: GPU Power comparison of the YOLO models on NVIDIA Xavier

The graph in figure 5.6 shows the Jetson Xavier GPU power comparison for four different versions of the YOLO.

The y-axis of the graph represents the power consumption in watts, and the x-axis represents the number of images processed per second. The graph shows that YOLOv4-tiny uses the least amount of power among the four models, followed by YOLOv3-tiny, YOLOv4, and YOLOv3, in that order.

In terms of power efficiency, YOLOv4-tiny is the most efficient among the four models, followed by YOLOv3-tiny, YOLOv4, and YOLOv3. This means that YOLOv4-tiny requires the least amount of power to process a given number of images, while YOLOv3 requires the most power.

Overall, the graph suggests that YOLOv4-tiny may be the best option for object detection tasks that require both high accuracy and low power consumption. However, if higher accuracy is required, YOLOv4 or YOLOv3 would be better options, but they would require more power. YOLOv3-tiny may be a good option for real-time processing with limited power resources.

6 Conclusion

Edge devices are becoming increasingly popular in various domains, such as automotive, healthcare, agricultural, smart cities, industrial automation, construction, and retail. These devices are being used to run state-of-the-art object detection models. To make these models run efficiently on edge devices, many software optimization techniques have been proposed.

In this thesis, we aim to investigate and characterize the performance of these software frameworks on edge devices. Through our analysis, we have discovered several interesting findings. Firstly, we have observed significant performance and accuracy improvements from the use of software optimizations. However, we have also noticed some unexpected behaviors, such as an increase in execution latency for the same model on more powerful hardware platforms.

Moreover, we conclude our study by discussing the implications of our findings for board power, GPU usage, and GPU power comparisons between two different edge devices - the NVIDIA Jetson TX2 and the NVIDIA Jetson Xavier NX. We have used different versions of the YOLO algorithm to compare these devices. Our findings provide insights into the balance between achieving high model accuracy and utilizing hardware resources, which can aid researchers and practitioners in choosing the most suitable edge device and software framework for their particular use case.

References

[1] D. Mishra, K. Rout, S. Mishra, S. R. Salkuti, and S. Reddy, "Various object detection algorithms and their comparison keywords: Conventional neural network decision review system region of interest single shot detector transfer learning," vol. 19, pp. 330–338, 01 2023.

[2] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," 2015.

[3] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, "Convolutional neural networks: an overview and application in radiology," pp. 611–629, 2018.

[4] A. John and D. D. Meva, "A comparative study of various object detection algorithms and performance analysis," *INTERNATIONAL JOURNAL OF COMPUTER SCIENCES AND ENGINEERING*, vol. 8, pp. 158–163, 10 2020.

[5] R. Gandhi, "R-cnn, fast r-cnn, faster r-cnn, yolo — object detection algorithms." [Online]. Available: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

[6] R. G. J. S. Shaoqing Ren, Kaiming He, "Faster r-cnn: Towards real-time object detection with region proposal networks." [Online]. Available: https://arxiv.org/abs/1506.01497

[7] Y. X. B. A. T. K. H. K. Ramyad Hadidi, Jiashen Cao, "Characterizing the deployment of deep neural networks on commercial edge devices." [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9041955

[8] www.accenture.com, "Edge computing." [Online]. Available: https://www.accenture.com/us-en/insights/cloud/edge-computing-index

[9] A. A. Süzen, B. Duman, and B. Şen, "Benchmark analysis of jetson tx2, jetson nano and raspberry pi using deep-cnn," in *2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*, 2020, pp. 1–5.

[10] H. Feng, G. Mu, S. Zhong, P. Zhang, and T. Yuan, “Benchmark analysis of yolo performance on edge intelligence devices,” in *2021 Cross Strait Radio Science and Wireless Technology Conference (CSRSWTC)*, 2021, pp. 319–321.

[11] R. Hadidi, J. Cao, Y. Xie, B. Asgari, T. Krishna, and H. Kim, “Characterizing the deployment of deep neural networks on commercial edge devices,” in *2019 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2019, pp. 35–48.

[12] I. R. G. F. J. H. G. T. P. P. C. D. R. C. L. Z. F. P. D. F. Tsung-Yi Lin Google Brain Genevieve Patterson MSR, Trash TV Matteo R. Ronchi Caltech Yin Cui Google Michael Maire TTI-Chicago Serge Belongie Cornell Tech Lubomir Bourdev WaveOne, “Coco dataset.” [Online]. Available: https://cocodataset.org/#home

[13] J. Redmon and A. Farhadi, “Yolov3: An incremental improvement,” 2018.

[14] A. A. Süzen, B. Duman, and B. Şen, “Benchmark analysis of jetson tx2, jetson nano and raspberry pi using deep-cnn,” *2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*, pp. 1–5, 2020.

[15] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

Acknowledgments

I am grateful to Professor Michele Rossi for providing me with the opportunity to conduct my research at the University of Padua, and for all of the resources and support they provided. I would also like to thank the Ph.D. students Aria Khoshsirat and Neda Shalavi, for their support during my research process.