VELLORE INSTITUTE OF TECHNOLOGY

ANDHRA PRADESH – 522237

VIT-AP UNIVERSITY

INTERNSHIP REPORT CARRIED OUT AT

INDIAN SPACE RESEARCH ORGANISATION (ISRO)

LABORATORY FOR ELECTRO-OPTICS SYSTEMS (LEOS)

- BANGALORE – DEPARTMENT OF SPACE, GOVT. OF INDIA

REPORT ON

"QUANTIZATION AWARE TRAINING FOR YOLOV5"

PROJECT REPORT SUBMITTED FOR THE PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF BACHELOR OF TECHNOLOGY

BACHELOR OF TECHNOLOGY (B. TECH)

[ELECTRONICS AND COMMUNICATION ENGINEERING]

BY,

BADDA ABHISHEK SUDARSHAN (21BEC7123)

GAURAV SHREEHARI G (21BEC7077)

D A SAI SANJANA T (21BEC7122)

ANUSRIRAJE R (21BEC7034)

UNDER THE SUPERVISION OF,

SHRI. PRAYANSHU SHARMA

(SSHS/SSSG/SSA)

ELECTRO-OPTICS LABORATORY

INDIAN SPACE RESEARCH ORGANIZATION

Vellore Institute of Technology – Andhra Pradesh

(Sate Private University Under The AP State Private Universities (Establishment and Regulation) Act, 2016)

Amaravati, Andhra Pradesh – 522 237, India, Web : www.vitap.ac.in

---

Ref:VIT-AP/AR/2022/3389 **06.05.2023**

TO WHOMSOEVER IT MAY CONCERN

This is to certify that **Mr. BADDA ABHISHEK SUDARSHAN (21BEC7123), Mr. GAURAV SHREEHARI G (21BEC7077), Ms. D A SAI SANJANA T (21BEC7122), Ms. ANUSRIRAJE R (21BEC70340** are a bonafide students of the Four - year B.Tech. Degree programme in Electronics and Communication Engineering under the School of Electronics Engineering of the Institute.

This certificate is issued in support of **his/her** application for "**Internship At Indian Space Research Organization (ISRO), LEOS in Bangalore**"

Agreed and countersigned by

ASST. DIRECTOR (SRS) **REGISTRAR**

ACKNOWLEDGMENT

We take this opportunity to acknowledge all those who have helped us in the completion of our internships.

Firstly, we express our sincere gratitude to the Laboratory for Electro-Optics Systems (LEOS), ISRO for giving us the opportunity to carry out the internship for 45 days at their premises.

We extend our gratitude and appreciation to **Shri. V. V. Ramana Reddy (Head, PPED)** who was instrumental in giving us this opportunity, and we thank him for all his support.

Next, we thank **Shri. Prayanshu Sharma (SSHS/SSSG/SSA)**, who was our internship guide, for his support, guidance, and supervision throughout the duration of our internships. We also thank all the Scientists/Engineers, LEOS, for their guidance and for sharing their experience and knowledge, as well as the rest of the staff for their support and guidance.

We thank **Smt. Rajeswari, Scientist/Engineer -SG, SH-SMCS1/SMCD1, SCG, ICA, URSC** and **Shri. Mahesh Kumar S**, **Scientist/Engineer – SF, Deputy Director, Office of Media & Public relations, (Head Quarters),** who supervised and involved us in working on the project, etc.

We thank the entire LEOS staff for being very supportive and helpful. The internship really helped our personal and professional development.

We would also like to express our gratitude to the LEOS Library and its staff for their continuous support during the time we were present there while completing our work.

Lastly, we thank our college, VIT – AP University, our parents, and our friends for their support and motivation at all times.

TABLE OF CONTENTS

TABLE OF FIGURES

CHAPTER - 1

1. INTRODUCTION

1.1.EXPLORING QUANTIZATION AWARE TRAINING IN YOLOV5

Quantization Aware Training (QAT) [1] stands at the forefront of deep learning techniques, offering a pivotal solution for optimizing the deployment of neural networks on resource-constrained platforms while preserving performance. In this era of breakthroughs and advancements in object detection, YOLO (You Only Look Once) models, including YOLOv5 and the ongoing evolution with YOLOv7, have reshaped the landscape of real-time object detection. **YOLO Paper [2]** This abstract provides an insight into the symbiotic relationship between Quantization Aware Training and YOLOv5.

The essence of Quantization Aware Training lies in recognizing that deep learning models often carry an excessive degree of precision in their weights. Through the process of quantization, these high-precision weights are transformed into leaner, lower bit-width representations. **[3]** YOLOv5, celebrated for its lightweight architecture, is poised to further elevate its efficiency through the incorporation of quantization techniques.

This study embarks on a journey to explore the intricacies of applying Quantization Aware Training to YOLOv5. It delves into the fine art of quantization, selection of the most fitting quantization schemes, and the delicate task of fine-tuning the model for optimal accuracy. The crux of the challenge is to retain exceptional object detection performance while operating with lower-precision weights.

The narrative doesn't stop at the intersection of QAT and YOLOv5. It stretches back through the evolution of object detection, showcasing the pivotal role of models like YOLO and YOLOv5 in transforming object detection into a real-time, single-stage paradigm, delivering efficiency and accuracy hand in hand. As the journey progresses to YOLOv7.

Object detection, **[4]** a fundamental task in computer vision, has permeated an array of applications, from autonomous vehicles to surveillance systems and industrial automation. YOLO models, with their fusion of efficiency and accuracy, have taken center stage. Quantization, on the other hand, offers the key to unlocking their full potential by adapting them to resource-constrained environments.

1.2. COMPUTER VISION

Computer vision [5] tasks include methods for acquiring, processing, analyzing and understanding digital images, and extraction of high-dimensional data from the real world in order to produce numerical or symbolic information, e.g., in the forms of decisions. Understanding in this context means the transformation of visual images (the input to the retina in the human analog) into descriptions of the world that make sense to thought processes and can elicit appropriate action. This image understanding can be seen as the disentangling of symbolic information from image data using models constructed with the aid of geometry, physics, statistics, and learning theory.

Sub-domains of computer vision include scene reconstruction, object detection, event detection, activity recognition, video tracking, object recognition, 3D pose estimation, learning, indexing, motion estimation, visual servoing, 3D scene modeling, and image restoration.

1.2.1. HOW IT WORKS

Two essential technologies are used to accomplish this: a type of machine learning called deep learning and a convolutional neural network (CNN). [6]

Machine learning uses algorithmic models that enable a computer to teach itself about the context of visual data. If enough data is fed through the model, the computer will "look" at the data and teach itself to tell one image from another. Algorithms enable the machine to learn by itself, rather than someone programming it to recognize an image.

A CNN helps a machine learning or deep learning model "look" by breaking images down into pixels that are given tags or labels. It uses the labels to perform convolutions (a mathematical operation on two functions to produce a third function) and makes predictions about what it is "seeing." The neural network runs convolutions and checks the accuracy of its predictions in a series of iterations until the predictions start to come true. It is then recognizing or seeing images in a way similar to humans.

1.3. CONVOLUTIONAL NEURAL NETWORK (CNN)

Convolutional Neural Network (CNN) is a type of Deep Learning neural network architecture commonly used in Computer Vision [7]. Computer vision is a field of Artificial Intelligence that enables a computer to understand and interpret the image or visual data.

When it comes to Machine Learning, Artificial Neural Networks perform really well. Neural Networks are used in various datasets like images, audio, and text. Different types of Neural Networks are used for different purposes, for example for predicting the sequence of words we use Recurrent Neural Networks more precisely an LSTM, similarly for image classification we use Convolution Neural networks. In this blog, we are going to build a basic building block for CNN.

The following sub-sections define all the architectural components of a CNN:

1) INPUT LAYER: Input layers introduce input data in the network, which normally represents an image structured as a data array of pixel values. Before feeding the image into a designed model, the spatial and channel dimensions of the input have to be reshaped according to the model or the used deep learning library specifications.

2) CONVOLUTIONAL LAYER: The core of a CNN is the convolutional layers. Learnable parameters in these layers form the kernels and the collection of the kernels in a layer is called a filter, which is subjected to a convolution multiplication through the full spatial depth of the input. A kernel defines the field of view on the convolution operation, whereas, as a tensor sample, a filter is the total number of kernels in a layer channel-wise. For instance, a filter with a size of "m × n × c" includes c kernels with "m × n" kernel dimension.

we assume the spatial dimensions of the input/output to be identical, which means that OD = ID, and the stride (i.e., the convolution kernel's step size) is 1. Then, the spatial 2D convolution outputs a tensor $O \in R^{ID \times ID \times F}$, computed as

$$O_{k,l,n} = \sum_{i,j,m} K_{i,j,m,n} I_{k+\hat{i},l+\hat{j},m},$$

where ˆi = i – [ID/2] and ˆj = j – [ID/2] denote the re-centred spatial indices; k, l and i, j are the indices over the spatial dimensions; whereas m, n provide indexation to the channels and filters.

3) PADDING OPERATION Padding is the placement of several extra pixel grids to the input's spatial plane to handle the output's spatial dimensions. In case of a demand for an equal dimension of input and output tensors, the padding operation could enable it. the output tensor's spatial dimensions are in the control of the padding operation. The effect of the padding operation in the output tensor's dimension (OD × OD) is defined by

$$OD = ID + 2P - KD + 1.$$

4) STRIDE OPERATION The stride value indicates that the filter, which is the weight tensor for the convolution process, slides on the input tensor in increments of one or more-pixel steps. This is another parameter that directly affects the output size. The dimension of the output tensor (OD × OD) is defined by

$$O_D = \frac{I_D + 2P - K_D}{S} + 1,$$

5) ACTIVATION FUNCTIONS: CNNs: use various activation functions, such as Rectified Linear Units (ReLU), Sigmoid, TanH, etc., to build the feature map, obtained through the convolution operation. These functions enable the introduction of nonlinearities to the layers, which increases learnability. In particular, ReLU has been frequently preferred in CNNs on the score of enabling several times faster training in comparison to the other activation functions.

6) POOLING OPERATION: Pooling layers take small-sized rectangular blocks, defined by KP, from the output feature map, and form subsamples from it to produce a single maximum, minimum, or average output from every block. After that, a new sampled feature map is formed, which is then used as an input for the next convolutional layer. By reducing the parameters of the feature map, the pooling layers allow to reduce the spatial size as well as the control of overfitting.

7) FULLY CONNECTED LAYER: The CNN architecture performs feature learning via the operations introduced above until the last convolutional and pooling layer output is derived.

The final CNN stage consists of FCLs and performs a high-level classification. It is placed after the last convolution or pooling operation has occurred. The flattening operation provides the transition from convolutional later to FCL by converting the data to a 1-dimensional array. Essentially, in this stage of the architecture, a usual neural network process for the operation of classification takes place.

8) OUTPUT LAYER: This is the final stage, and the predicted result for the input image is obtained via different cost functions. For instance, Softmax multi-class classifier is deployed. Thus, a cost is produced for a prediction by comparing the predicted result and the real data from the training set.

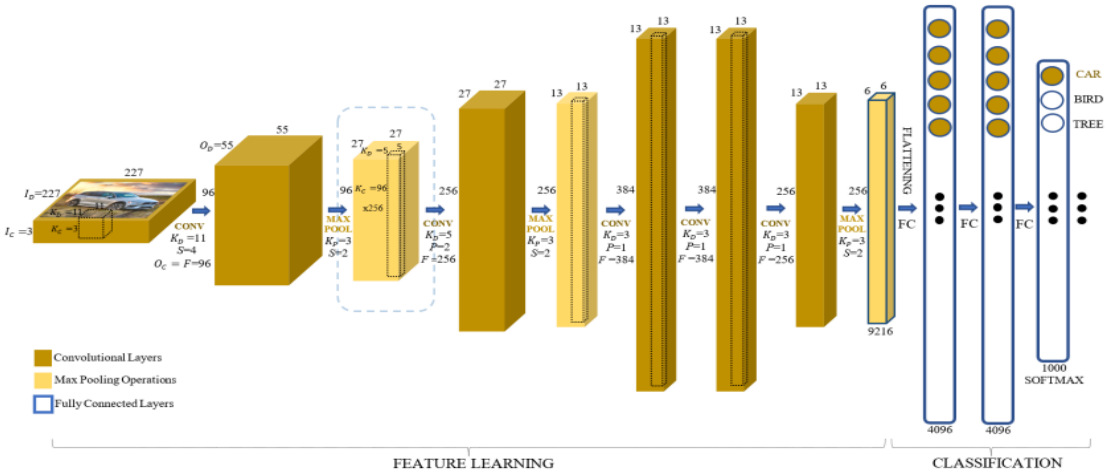


Figure 1:CNN ARCHITECTURE

(https://www.researchgate.net/publication/358202576_Towards_Performing_Image_Classification_and_Object_Detection_with_Convolutional_Neural_Networks_in_Autonomous_Driving_Systems_A_Survey_December_2021)

1.3.1. HOW CONVOLUTIONAL LAYERS WORKS

Convolution Neural Networks or covnets are neural networks that share their parameters. Imagine you have an image. It can be represented as a cuboid having its length, width (dimension of the image), and height (i.e. the channel as images generally has red, green, and blue channels). **[8]**

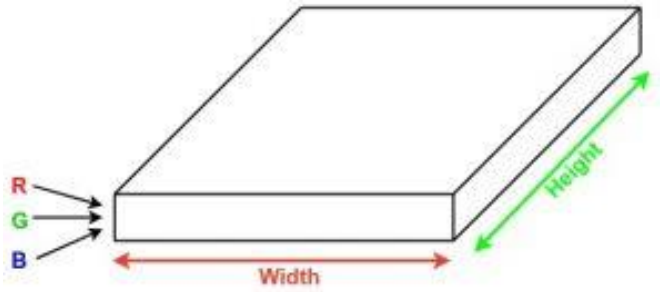


Figure 2: Representing Cuboid having length, width and height

(https://www.geeksforgeeks.org/introduction-convolution-neural-network/)

Now imagine taking a small patch of this image and running a small neural network, called a filter or kernel on it, with say, K outputs and representing them vertically. Now slide that neural network across the whole image, as a result, we will get another image with different widths, heights, and depths. Instead of just R, G, and B channels now we have more channels but lesser width and height. This operation is called Convolution. If the patch size is the same as that of the image it will be a regular neural network. Because of this small patch, we have fewer weights.

- Convolution layers consist of a set of learnable filters (or kernels) having small widths and heights and the same depth as that of input volume (3 if the input layer is image input).
- For example, if we have to run convolution on an image with dimensions 34x34x3. The possible size of filters can be axax3, where 'a' can be anything like 3, 5, or 7 but smaller as compared to the image dimension.
- During the forward pass, we slide each filter across the whole input volume step by step where each step is called stride (which can have a value of 2, 3, or even 4 for high-

dimensional images) and compute the dot product between the kernel weights and patch from input volume.

- As we slide our filters, we'll get a 2-D output for each filter and we'll stack them together as a result, we'll get output volume having a depth equal to the number of filters. The network will learn all the filters.

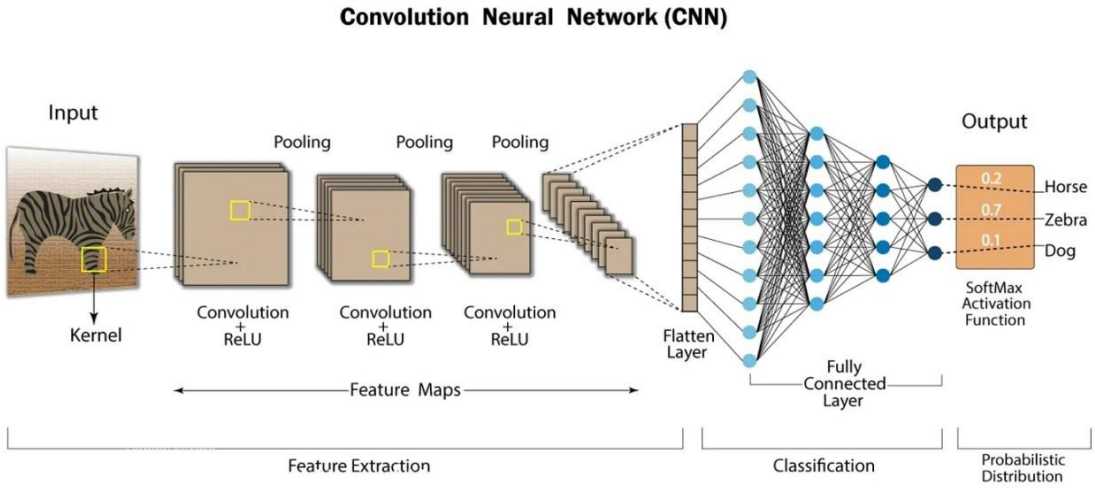


Figure 3: CNN Layers

(https://www.analyticsvidhya.com/blog/2022/03/basics-of-cnn-in-deep-learning/)

CHAPTER – 2

2. OBJECT DETECTION

Object detection [9] is a computer technology related to computer vision and image processing that deals with detecting instances of semantic objects of a certain class (such as humans, buildings, or cars) in digital images and videos. Well-researched domains of object detection include face detection and pedestrian detection. Object detection has applications in many areas of computer vision, including image retrieval and video surveillance.

Real-time object detection **[10]** is a very important topic in computer vision, as it is often a necessary component in computer vision systems. For example, multi-object tracking, autonomous driving, robotics, medical image analysis, etc. The computing devices that execute real-time object detection is usually some mobile CPU or GPU, as well as various neural processing units (NPU) developed by major manufacturers.

- Every object class has its own special features that help in classifying the class – for example all circles are round. Object class detection uses these special features. For example, when looking for circles, objects that are at a particular distance from a point (i.e., the center) are sought. Similarly, when looking for squares, objects that are perpendicular at corners and have equal side lengths are needed.

Currently state-of-the-art real-time object detectors are mainly based on YOLO and FCOS, which are being able to become a state-of-the-art real-time object detector usually requires the following characteristics: (1) a faster and stronger network architecture; (2) a more effective feature integration method; (3) a more accurate detection method; (4) a more robust loss function; (5) a more efficient label assignment method; and (6) a more efficient training method.

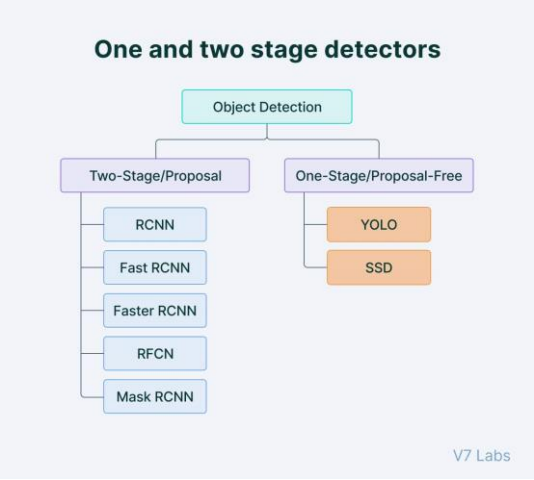


Figure 4: One & Two detectors

(https://www.v7labs.com/blog/object-detection-guide)

2.1. TYPES OF DETECTORS

1) SINGLE SHOT DETECTORS (SSD):

Single Shot Detectors (SSD) **SSD Paper [11]** is a popular real-time object detection algorithm known for its speed and accuracy. It belongs to the family of convolutional neural network (CNN)-based object detection methods. SSD is designed to efficiently perform both object localization and classification within a single pass of the network, making it well-suited for real-time applications.

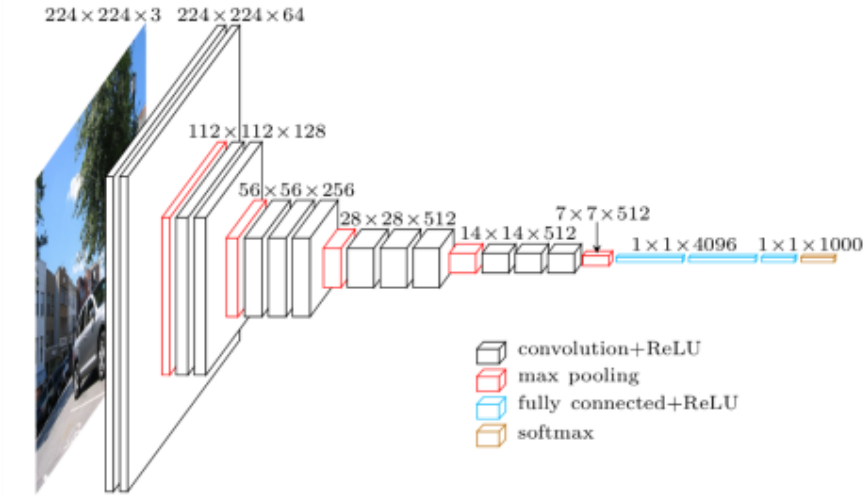


Figure 5: Single Shot Detection (SSD) Algorithm

(https://towardsdatascience.com/understand-the-architecture-of-cnn-90a25e244c7)

REASONS BEHIND CONSIDERING THEM:

- **Real-Time Performance:** SSD is known for its real-time object detection capabilities, making it suitable for applications that require swift object detection, such as video analysis or robotics.
- **Multi-Scale Detection:** SSD excels at handling objects of various sizes within a single network architecture, which is crucial for detecting objects at different distances from the camera.
- **Efficiency:** SSD's efficient design, which combines object localization and classification in a single pass, makes it computationally efficient, allowing it to run on resource-constrained devices.

2) R-CNN (REGIONS WITH CONVOLUTIONAL NEURAL NETWORKS):

R-CNN, which stands for "Regions with Convolutional Neural Networks," **R-CNN Paper [12]** is an early and influential object detection framework in computer vision. Developed by Ross Girshick, et al., R-CNN was introduced as a solution to the problem of object localization and classification within images.

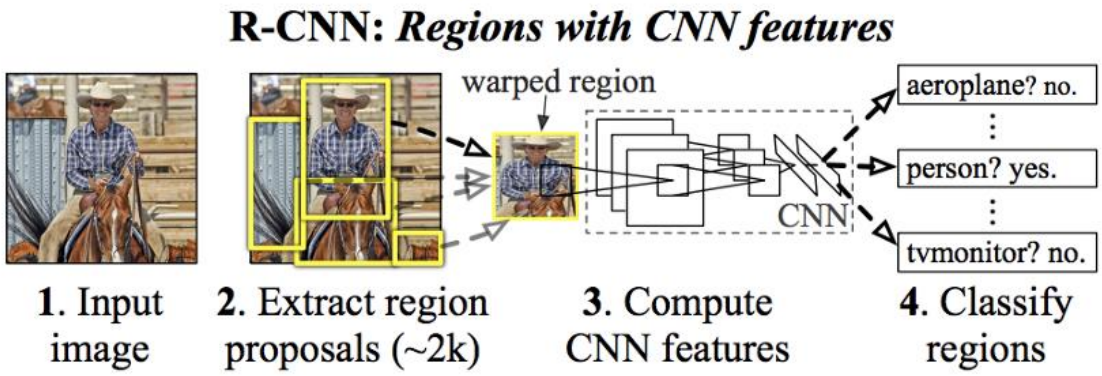


Figure 6: R-CNN

(https://paperswithcode.com/method/r-cnn)

REASONS BEHIND CONSIDERING THEM:

- **High Accuracy:** R-CNN variants, like Fast R-CNN and Faster R-CNN, are known for their high accuracy in object detection tasks. If your project demands precise object localization and classification, they might be a suitable choice.
- **Localization Refinement:** R-CNN models refine object localization through bounding box regression, which can lead to accurate object boundaries.
- **Object Recognition:** R-CNN-based models can not only detect objects but also recognize their classes, which is valuable in many applications.

CHAPTER - 3

3. QUANTIZATION

Quantization [13] is a process of reducing the number of distinct values in a data set or signal. In the context of digital data, it involves mapping continuous or high-precision values to a smaller set of discrete, lower-precision values. This is done to make data more manageable, efficient, and suitable for storage, transmission, or computation.

For instance, in digital image processing, quantization might involve reducing the range of colors in an image to a smaller set of distinct colors. In digital signal processing, it can refer to converting a continuous signal into a series of discrete values.

In the context of machine learning and neural networks [14], quantization often refers to reducing the precision of numerical values. For example, weights and activations in a neural network, which are typically stored as high-precision floating-point numbers, can be quantized to lower-precision representations like integers or fixed-point numbers. This process reduces memory usage and computational requirements, which is particularly useful for deploying models on resource-constrained devices while maintaining acceptable accuracy.

It is a process of mapping continuous infinite values to a smaller set of discrete finite values. In the context of simulation and embedded computing [15], it is about approximating real-world values with a digital representation that introduces limits on the precision and range of a value.

Quantization optimizations can be made when the targeted hardware (GPU, FPGA, CPU) architecture is taken into consideration. This includes computing in integers, utilizing hardware accelerators, and fusing layers. The quantization step is an iterative process to achieve acceptable accuracy of the network.

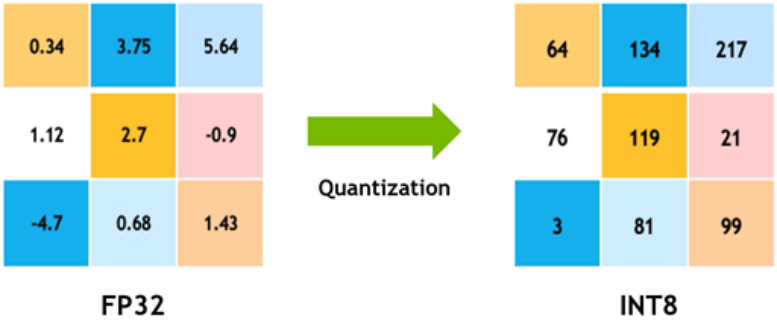


Figure 7: Achieving FP32accuracy for INT8 Inference

(https://developer.nvidia.com/blog/achieving-fp32-accuracy-for-int8-inference-using-quantization-aware-training-with-tensorrt/)

3.1. APLLICATIONS OF QUANTIZATION

Quantization is a fundamental concept in various fields of engineering and computer science, including digital signal processing, image and video compression, and audio encoding and compression. The process of quantization involves mapping a continuous range of values to a finite set of discrete values. This technique is used to reduce the amount of data required to represent a signal or image, making it easier to store, transmit, and process. **[16]**

DIGITAL SIGNAL PROCESSING:

Digital signal processing (DSP) is the use of digital processing techniques to manipulate and analyze signals. Quantization is a crucial component of DSP because it enables the conversion of analog signals, such as sound and video, into digital signals that can be processed by computers. In DSP, quantization is used to convert the continuous amplitude of a signal into a finite set of discrete values, which can then be manipulated using digital signal processing techniques. This process is essential in applications such as audio and video processing, where signals need to be analyzed and manipulated in real-time.

IMAGE AND VIDEO COMPRESSION:

Image and video compression are techniques used to reduce the size of image and video files while maintaining their visual quality. Quantization is a key component of these techniques, as it enables the reduction of the number of bits required to represent a signal. In image and video compression, vector quantization is often used to reduce the size of the file. This technique involves dividing the image or video into blocks and quantizing each block separately. The quantized values are then represented using a codebook, which allows for efficient storage and transmission of the compressed file.

AUDIO ENCODING AND COMPRESSION:

Audio encoding and compression are techniques used to reduce the size of audio files while minimizing the loss of quality. Quantization is a critical component of these techniques, as it enables the reduction of the number of bits required to represent an audio signal. Differential quantization is often used in audio compression due to its effectiveness in reducing quantization error. This technique involves quantizing the difference between consecutive samples, rather than the samples themselves.

3.2. TYPES OF QUANTIZATION

➢ POST TRAINING QUANTIZATION (PTQ):

Post Training Quantization **[17]** computes *scale* after network has been trained. A representative dataset is used to capture the distribution of activations for each activation tensor, then this distribution data is used to compute the *scale* value for each tensor.

TensorRT provides a workflow for PTQ, called *calibration*.

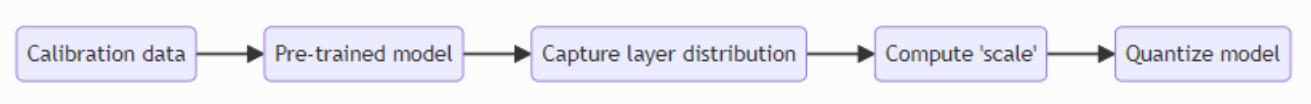


Figure 8: Post Training Quantization

(https://docs.nvidia.com/deeplearning/tensorrt/tensorflow-quantization-toolkit/docs/docs/qat.html)

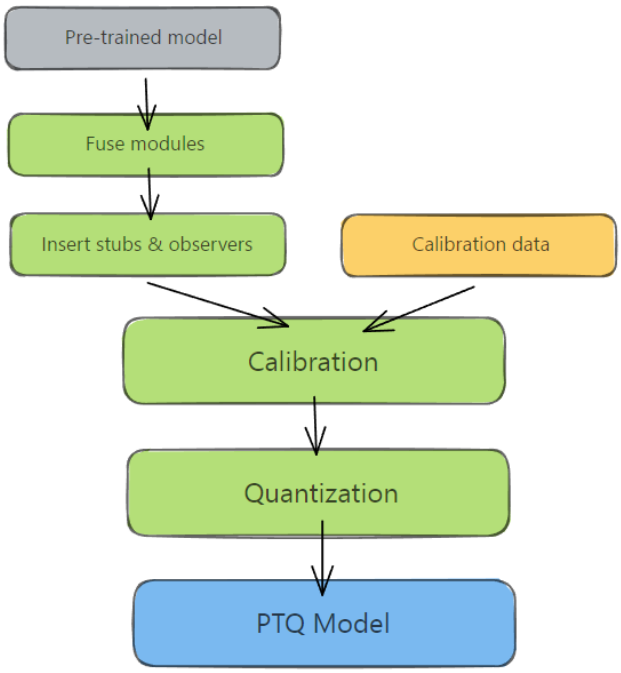


Figure 9: Steps in Post-Training Static Quantization

(https://pytorch.org/blog/quantization-in-practice/#:~:text=Steps%20in%20Post%2DTraining%20Static%20Quantization)

(+) Static quantization has faster inference than dynamic quantization because it eliminates the float<->int conversion costs between layers. **[18]**

(-) Static quantized models may need regular re-calibration to stay robust against distribution-drift.

➢ QUANTIZATION AWARE TRAINING (QAT):

Quantization Aware Training **[19]** aims at computing scale factors during training. Once the network is fully trained, Quantize (Q) and Dequantize (DQ) nodes are inserted into the graph following a specific set of rules. The network is then further trained for few epochs in a process called *Fine-Tuning*. Q/DQ nodes simulate quantization loss and add it to the training loss during fine-tuning, making the network more resilient to quantization. In other words, QAT is able to better preserve accuracy when compared to PTQ.

Figure 10: Quantization Aware Training

(https://docs.nvidia.com/deeplearning/tensorrt/tensorflow-quantization-toolkit/docs/docs/qat.html)

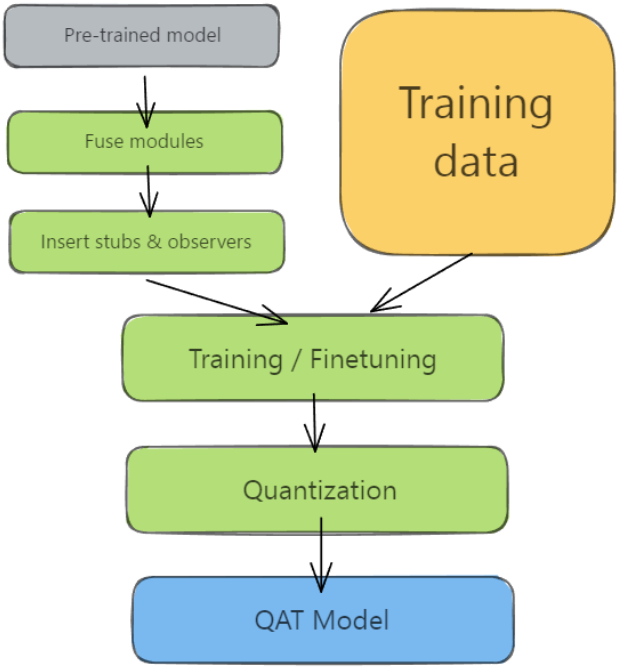


Figure 11: Steps in Quantization-Aware Training

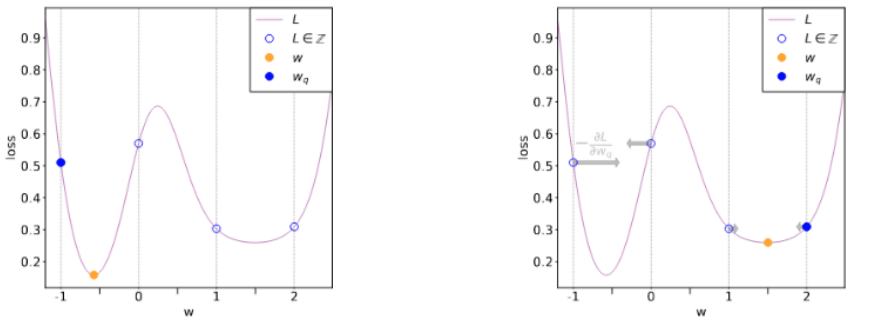
(https://pytorch.org/blog/quantization-in-practice/#:~:text=Steps%20in%20Quantization%2DAware%20Training)

(+) QAT yields higher accuracies than PTQ. **[20]**

(+) Qparams can be learned during model training for more fine-grained accuracy

(-) Computational cost of retraining a model in QAT can be several hundred epochs.

- The PTQ approach is great for large models, but accuracy suffers in smaller models. This is of course due to the loss in numerical precision when adapting a model from FP32 to the INT8 realm. QAT tackles this by including this quantization error in the training loss, thereby training an INT8-first model.



(a) Post training quantization (b) After quantization aware fine-tuning

Figure 12: Comparison of PTQ and QAT convergence

(https://pytorch.org/blog/quantization-in-practice/#:~:text=Comparison%20of%20PTQ%20and%20QAT%20convergence)

CHAPTER – 4

4. YOLO (YOU ONLY LOOK ONCE)

YOLO (You Only Look Once) [21] is a popular real-time object detection system that revolutionized the field of computer vision by offering a highly efficient and accurate approach to detect and locate objects in images and videos. YOLO processes images in a single forward pass through a neural network and directly predicts the bounding boxes and class probabilities for all objects present in the image.

4.1. HOW YOLO WORKS

YOLO works [22] by dividing an input image into a grid of cells and making predictions for each cell. For each cell, YOLO predicts multiple bounding boxes and the associated class probabilities. The key steps in how YOLO works are as follows:

1. **Input Image Division:** The input image is divided into an SxS grid. Each cell in the grid is responsible for predicting objects whose centers fall within that cell.
2. **Bounding Box Prediction:** For each cell, YOLO predicts multiple bounding boxes, often referred to as anchor boxes. These bounding boxes consist of coordinates (x, y) for the box's center, width (w), and height (h). YOLO also predicts a confidence score for each bounding box, indicating the likelihood that the box contains an object.
3. **Class Prediction:** YOLO predicts the class probabilities for each bounding box in each cell. It determines the object class associated with each box (e.g., car, person, cat, etc.).
4. **Confidence Thresholding:** YOLO applies a confidence threshold to filter out low-confidence predictions. Bounding boxes with confidence scores below the threshold are discarded.
5. **Non-Maximum Suppression:** YOLO uses non-maximum suppression to eliminate duplicate or highly overlapping bounding box predictions. It keeps only the most confident prediction for each object, removing redundant boxes.

4.2. BASIC ARCHITECTURE

The basic architecture of YOLO [23] consists of several convolutional layers followed by fully connected layers for prediction. YOLO versions (e.g., YOLOv1, YOLOv2, YOLOv3, YOLOv4, YOLOv5) differ in network depth and design, but the core architecture remains consistent. A common structure includes:

- **Input Layer:** Accepts an image as input.
- **Convolutional Layers:** A series of convolutional layers extract features from the image at multiple scales.
- **Fully Connected Layers:** These layers make predictions for bounding boxes, class probabilities, and confidence scores.
- **Output Layer:** The output layer provides the final predictions, including bounding box coordinates, class probabilities, and confidence scores.

YOLO's single-pass architecture allows it to achieve real-time object detection on a variety of platforms, making it suitable for applications such as autonomous driving, surveillance, and object tracking. Its efficiency and accuracy have contributed to its popularity in the computer vision community.

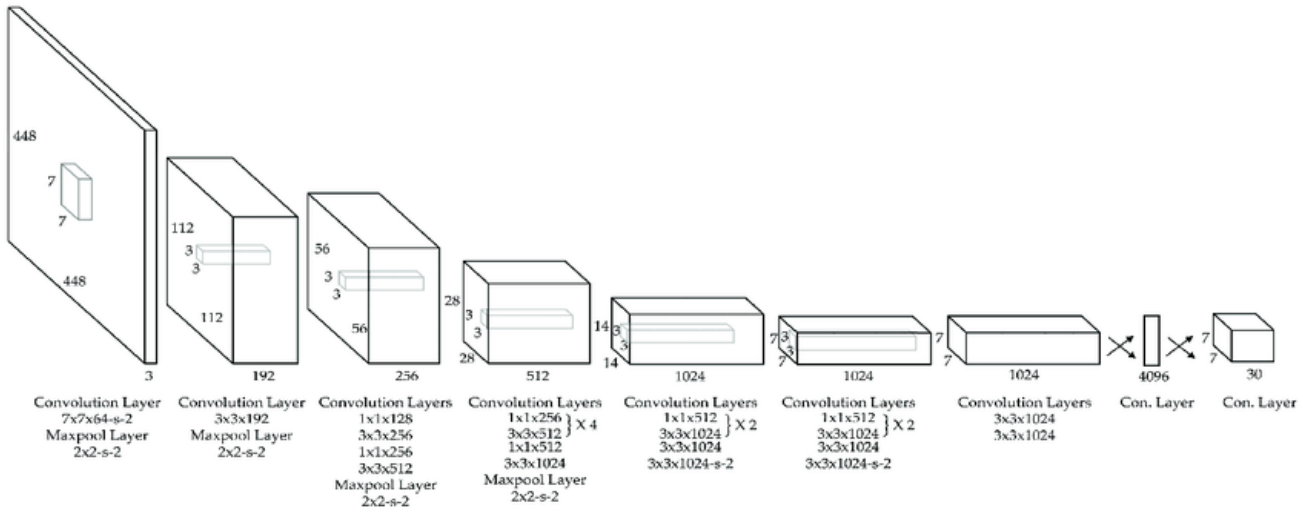


Figure 13: YOLO ARCHITECTURE

(https://www.researchgate.net/figure/YOLO-network-architecture-adapted-from-44_fig1_330484322)

CHAPTER – 5

5. PYTORCH

PyTorch [24] is a machine learning framework based on the Torch library, used for applications such as computer vision and natural language processing, originally developed by Meta AI and now part of the Linux Foundation umbrella. It is free and open-source software released under the modified BSD license. Although the Python interface is more polished and the primary focus of development, PyTorch also has a C++ interface.

A number of pieces of deep learning software are built on top of PyTorch, including Tesla Autopilot, Uber's Pyro, Hugging Face's Transformers, PyTorch Lightning, and Catalyst.

PyTorch provides two high-level features: [25]

- Tensor computing (like NumPy) with strong acceleration via graphics processing units (GPU)
- Deep neural networks built on a tape-based automatic differentiation system.

5.1. PYTORCH IN YOLOV5

PyTorch plays a central role in YOLOv5 [26], a state-of-the-art object detection model. YOLOv5 is built on the PyTorch framework, leveraging its capabilities for defining the model architecture, loading and preprocessing data, training the model, and performing real-time inference. Developers can harness PyTorch's flexibility to customize the YOLOv5 model for specific object detection tasks and easily deploy it across various platforms, making it a versatile and widely adopted solution in computer vision applications.

5.2. HOW PYTORCH IS USED IN YOLOV5 [27]

1. **Model Architecture:** YOLOv5's architecture, including the neural network layers, loss functions, and optimization algorithms, is defined using PyTorch. This allows developers to design and customize the model's structure and parameters within the PyTorch framework.
2. **Data Loading:** PyTorch's data loading utilities, such as **DataLoader**, are used to efficiently load and preprocess datasets for training and inference. YOLOv5 typically works with image datasets and annotations for object detection tasks.
3. **Training:** YOLOv5 can be trained using PyTorch's training loop and loss functions. During training, PyTorch handles backpropagation, parameter updates, and gradient computation. Users can leverage PyTorch's flexibility to fine-tune the model on their specific datasets and objectives.
4. **Inference:** For object detection in real-time or batch inference, PyTorch is used to load the trained YOLOv5 model and apply it to input images or videos. This involves processing the input data through the network and post-processing the output predictions, including non-maximum suppression and drawing bounding boxes.
5. **Model Deployment:** PyTorch allows YOLOv5 models to be deployed on various platforms, including edge devices, cloud servers, and mobile applications, making it accessible for a wide range of applications.

CHAPTER – 6

6. QUANTIZATION AWARE TRAINING (QAT) FOR YOLOV5

Quantization Aware Training (QAT) for YOLOv5 [28] is a technique that enhances the performance of the YOLOv5 object detection model, ensuring its effectiveness when deployed on resource-constrained platforms. YOLOv5 is renowned for its real-time object detection capabilities, and QAT fine-tunes the model to be "quantization-aware."

In this process, simulated quantization effects are introduced during the model's training phase, allowing it to adapt to the lower-precision numerical values used during inference. Specifically, quantization layers are incorporated into the YOLOv5 architecture, simulating the quantization of weights and activations that will occur during real-world usage. The loss function used in training is modified to encourage the model to make predictions as if it were quantized, and gradients are computed with quantization in mind.

The result is a YOLOv5 model that seamlessly maintains its accuracy and performance when subjected to the reduced precision of quantization. This makes YOLOv5 an excellent choice for real-time object detection applications on edge devices, mobile platforms, and embedded systems, where computational efficiency and memory optimization are essential. QAT for YOLOv5 ensures that the model remains a powerful and practical tool for a wide array of computer vision applications.

6.1. QAT STEPS FOR YOLOV5:

[29]

1. **Quantization Layers:** During training, additional quantization layers are introduced, simulating the quantization of weights and activations as they would occur during inference. These layers mimic the quantization process and allow for gradients to flow through the network.
2. **Loss Function:** The loss function used for QAT includes terms that encourage the model to make predictions and updates as if it were quantized. This ensures that the model learns to operate effectively with the reduced precision.

3. **Gradients:** Gradients are computed as if the model is quantized, which helps to maintain accuracy when lower-precision values are used during inference. This process fine-tunes the model to be quantization-aware.

Quantization Aware Training (QAT) for YOLOv5 is particularly relevant in the context of real-time object detection. It bridges the gap between the high-precision demands of deep learning models and the need for efficient deployment on hardware with limited computational resources. This is especially crucial in applications like autonomous vehicles, surveillance systems, and robotics, where rapid decision-making based on sensor data is imperative.

By incorporating quantization into the training process, QAT ensures that YOLOv5 is well-prepared for deployment in the real world. It is instrumental for scenarios that require a balance between the model's memory efficiency and its capacity to deliver accurate object detection results in real-time.

This combination of YOLOv5 and QAT provides a powerful solution for various domains, including healthcare, retail, security, and more. Whether it's for detecting anomalies in medical images, counting objects on store shelves, or identifying objects of interest in surveillance footage, YOLOv5 with QAT empowers developers to create applications that demand both accuracy and efficiency, making it a valuable asset in the field of computer vision.

6.1.1. STEPS TO PERFORM QAT FOR YOLOV5

Quantization Aware Training (QAT) for YOLOv5 is a process of training the YOLOv5 object detection model while considering the effects of quantization that will occur during deployment. Here are the steps to perform QAT for YOLOv5: **[30]**

1. **Prepare Your Data:** Start by preparing your object detection dataset. This typically includes a labeled dataset with images and annotations indicating the objects and their locations within the images. Ensure that your dataset is well-structured and properly annotated.
2. **Choose or Build the YOLOv5 Model:** Select the YOLOv5 model architecture that you intend to use. You can choose from different variants of YOLOv5 depending on your specific requirements and hardware constraints.
3. **Integrate Quantization Layers:** Modify the YOLOv5 architecture by adding quantization layers that simulate the quantization effects. These layers will be used during training to prepare the model for lower-precision values during inference.
4. **Adjust the Loss Function:** Modify the loss function used for training to account for the quantization effects. Ensure that the loss function encourages the model to make predictions as if it were quantized.
5. **Incorporate Quantization-Aware Gradients:** During backpropagation, compute gradients as if the model is already quantized. This fine-tunes the model to be quantization-aware, allowing it to adapt to lower-precision values.
6. **Training:** Train the modified YOLOv5 model on your prepared dataset. During the training process, the model will learn to make predictions and updates considering the quantization effects.
7. **Validation and Fine-Tuning:** After training, validate the model's performance to ensure that it maintains the desired level of accuracy. If necessary, fine-tune the model to recover any accuracy lost during the quantization-aware training process.
8. **Inference:** Once the model is trained and fine-tuned, it is ready for deployment. During inference, the quantization-aware YOLOv5 model will operate effectively with lower-precision values while still providing accurate object detection results.

6.1.2. HOW IS IT PERFORMED?

[31] We first decide on a scheme for quantization. It means deciding what factors we want to include to convert the float values to lower precision integer values with minimum loss of information. In this article, we will be using the quantization scheme used in as a reference. So, we introduce two new parameters for this purpose: scale and zero-point. As the name suggests scale parameter is used to scale back the low-precision values back to the floating-point values. It is stored in full precision for better accuracy. On the other hand, zero-point is a low precision value that represents the quantized value that will represent the real value 0. The advantage of zero-point is that we can have a wider range for integer values even for skewed tensors. So real values (r) could be derived from quantized values (q) in the following way:

$$r = S(q - Z)$$

Equation 1

Here S and Z represent scale and zero-point respectively.

6.2. FAKE QUANTIZATION

Fake quantization [32], often encountered in the context of Quantization Aware Training (QAT) for deep learning models, is a technique used to simulate the effects of quantization during the training process. It's sometimes called "fake" quantization because it doesn't involve actual quantization (conversion of high-precision values to lower-precision representations) but rather approximates the behavior of quantization.

The primary purpose of fake quantization in QAT is to train the model to account for the precision loss that will occur when the model is deployed with lower-precision weights and activations. Here's how fake quantization works in QAT:

1. **Quantization Simulation:** During QAT, fake quantization is introduced by adding quantization functions or layers to the neural network. These functions simulate the quantization process without actually changing the precision of values.
2. **Modified Forward Pass:** During forward pass, data flows through these fake quantization layers. These layers mimic the behavior of quantization by quantizing the data for the

purpose of computation, but the results are not actually converted to lower precision. This allows the model to learn to operate under the assumption that quantization will take place.

3. **Gradient Calculation:** During backpropagation (backward pass), gradients are calculated as if the model's weights and activations have undergone real quantization. This means that the gradients take into account the quantization effects, which helps the model adapt to the expected precision loss.

4. **Training with Quantization Effects:** Throughout the training process, the model makes predictions and updates as if it were operating with quantized values. This fine-tunes the model to be robust and accurate when deployed with lower-precision numerical representations.

Fake quantization serves as a crucial bridge between high-precision training and lower-precision deployment. It ensures that deep learning models trained with QAT are capable of maintaining their performance and accuracy in real-world scenarios where quantization is applied.

6.3. FAKEQUANT [33]

We introduce something known as FakeQuant nodes into our model after every operation involving computations to obtain the output in the range of our required precision. A FakeQuant node is basically a combination of Quantize and Dequantize operations stacked together.

6.4. QUANTIZE OPERATION [34]

Its main role is to bring the float values of a tensor to low precision integer values. It is done based on the above-discussed quantization scheme. Scale and zero-point are calculated in the following way:

The main role of scale is to map the lowest and highest value in the floating range to the highest and lowest value in the quantized range. In the case of 8-bit quantization, the quantized range would be [-128,127].

$$scale = \frac{f_{max} - f_{min}}{q_{max} - q_{min}}$$

Equation 2

here f_{max} and f_{min} represent the maximum and minimum value in floating-point precision, q_{max} and q_{min} represent the maximum value and minimum value in the quantized range.

Similarly, we can find the zero-point by establishing a linear relationship between the extreme floating-point values and the quantized values.

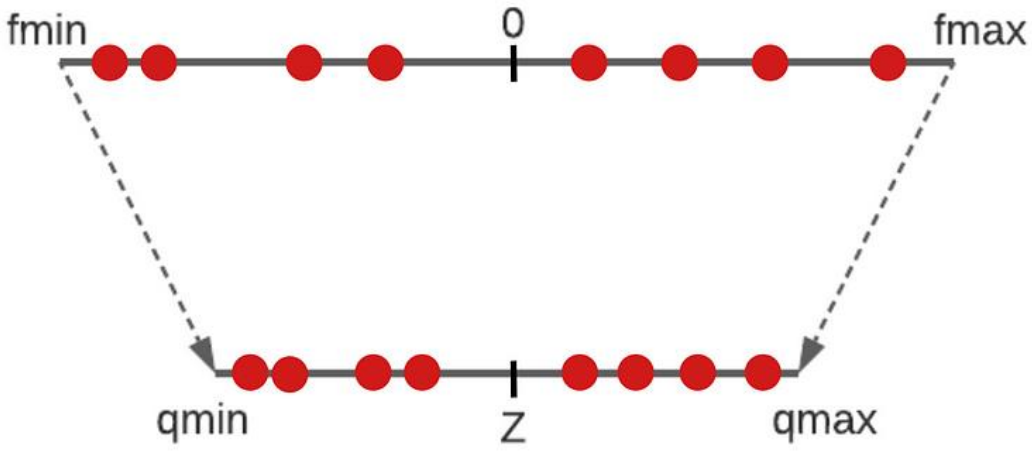


Figure 14: Representation of scaling from floating-point domain to quantized domain

(https://towardsdatascience.com/inside-quantization-aware-training-4f91c8837ead#:~:text=the%20quantized%20values.-,Figure%201,-.%20Representation%20of%20scaling)

Considering we have coordinates of two points of a straight line (q_{mi}, f_{min}) and (q_{max},f_{max}), we can obtain its equation in the form of y = mx +c, x being the quantized values and y being the real values. So, to get the mapping of 0 in the quantized domain, we just find the value of x with y=0. On solving this we would get:

$$Z = q_{min} - \frac{f_{min}}{scale}$$

Equation 3

But what if 0 doesn't lie between f_{max} and f_{min}, our Zero point would then go out of the quantization range. To overcome this, we can set Z to q_{max} or q_{min} depending on which side it lies on.

Now we have everything for our Quantize operation and we can obtain quantized values from floating-point values using the equation:

$$q = \left\lfloor \frac{r}{S} + Z \right\rceil$$

Equation 4

6.4.1. DEQUANTIZE OPERATION [35]

To obtain back the real values we put the quantized value in Equation 1, so that becomes:

$$r_{new} = S(\left\lfloor \frac{r}{S} + Z \right\rceil - Z)$$

Equation 5

6.5. CREATING A TRAINING GRAPH

Now that we have defined our FakeQuant nodes, we need to determine the correct position to insert them in the graph. We need to apply Quantize operations on our weights and activations using the following rules: **[36]**

- Weights need to be quantized before they are multiplied or convolved with the input.
- Our graph should display inference behavior while training so the BatchNorm layers must be folded and Dropouts must be removed. Details on BatchNorm folding can be found here.
- Outputs of each layer are generally quantized after the activation layer like Relu is applied to them which is beneficial because most optimized hardware generally have the activation function fused with the main operation.

- We also need to quantize the outputs of layers like Concat and Add where the outputs of several layers are merged.
- We do not need to quantize the bias during training as we would be using int32 bias during inference and that can be calculated later on with the parameters obtained using the quantization of weights and activations. We will discuss that later in this post.

Scales and Zero-points of weights are determined simply as discussed in the previous section. To determine scales and zero-points of activations we need to maintain an exponential moving average of the max and min values of the activation in the floating-point domain so that our parameters are smoothened over the data obtained from many images.

So the fake quantize operations are inserted in the graph as shown below.

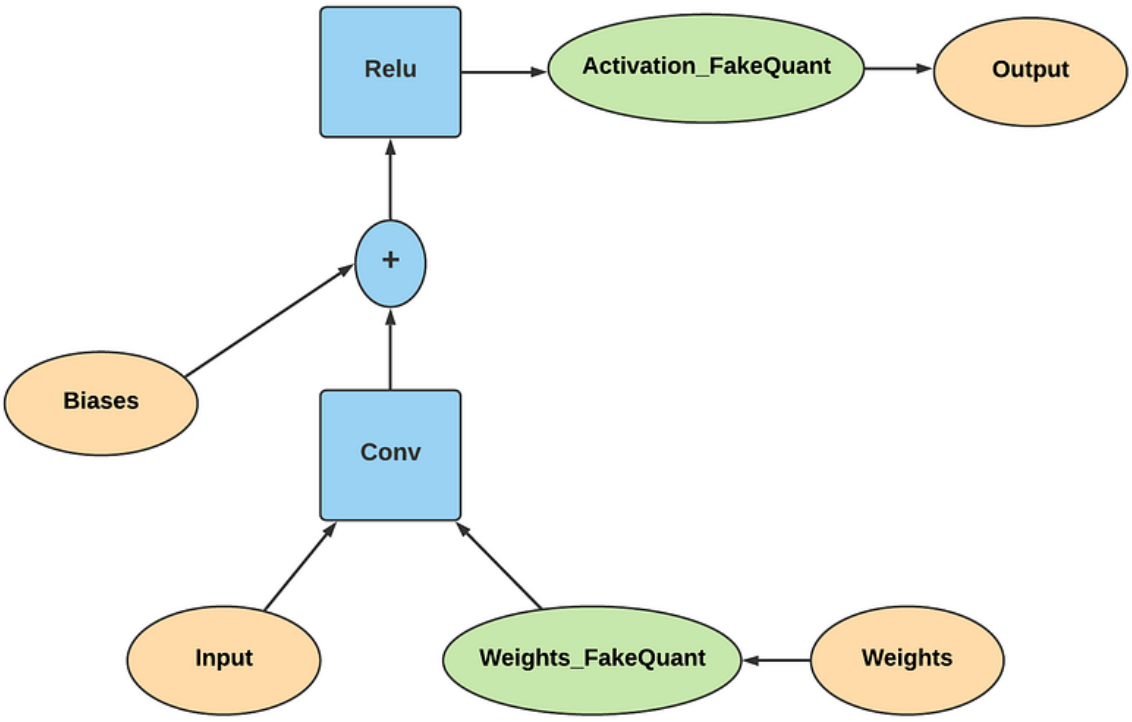


Figure 15: Visual Representation of a Quant-Aware training graph

(https://towardsdatascience.com/inside-quantization-aware-training-4f91c8837ead#:~:text=as%20shown%20below.-,Figure%202,-.%20Visual%20Representation%20of)

Now that our graph is ready, we need to prepare it for training. While training we have to simulate the quantization behavior only in the forward pass to induce the quantization error, the backward pass remains the same and only the floating-point weights are updated during training.

- All weights and biases are stored in FP32, and backpropagation happens as usual. However, in the forward pass, quantization is internally simulated via FakeQuantize modules. They are called fake because they quantize and immediately dequantize the data, adding quantization noise similar to what might be encountered during quantized inference. The final loss thus accounts for any expected quantization errors. Optimizing on this allows the model to identify a wider region in the loss function, and identify FP32 parameters such that quantizing them to INT8 does not significantly affect accuracy.

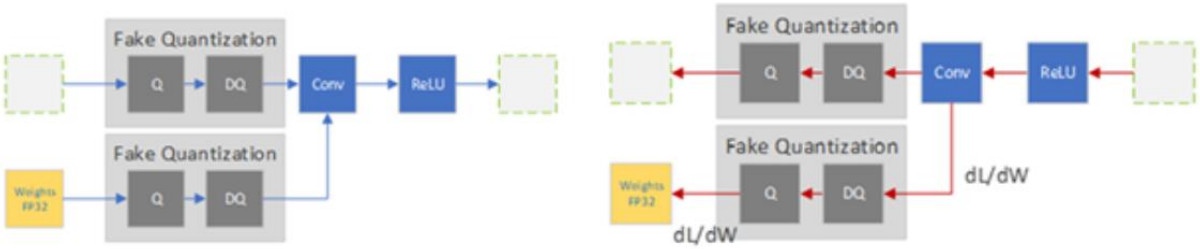


Figure 16: Fake Quantization in the forward and backward pass

(https://pytorch.org/blog/quantization-in-practice/#:~:text=significantly%20affect%20accuracy.-,Fig%207,-.%20Fake%20Quantization%20in)

- Not all layers respond to quantization equally, some are more sensitive to precision drops than others. Identifying the optimal combination of layers that minimizes accuracy drop is time-consuming, so suggest a one-at-a-time sensitivity analysis to identify which layers are most sensitive, and retaining FP32 precision on those.

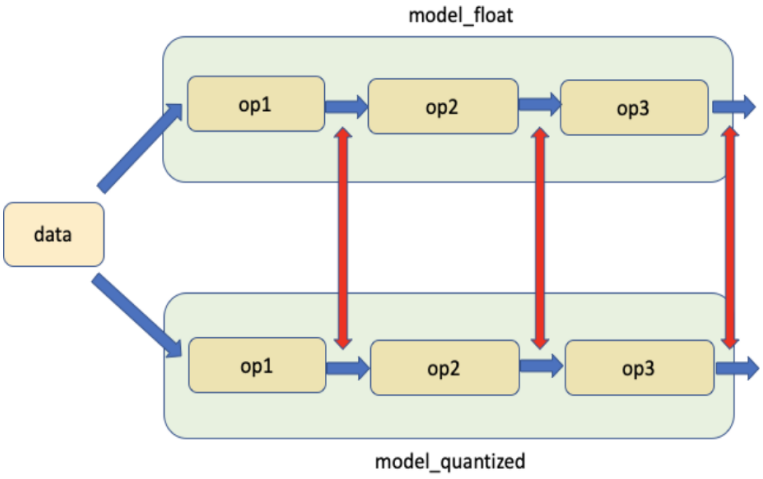


Figure 17: Comparing model weights and activations

(https://pytorch.org/blog/quantization-in-practice/#:~:text=guiding%20further%20optimizations.-,Fig%208,-.%20Comparing%20model%20weights)

6.6. QUANTIZATION WORK FLOW [37]

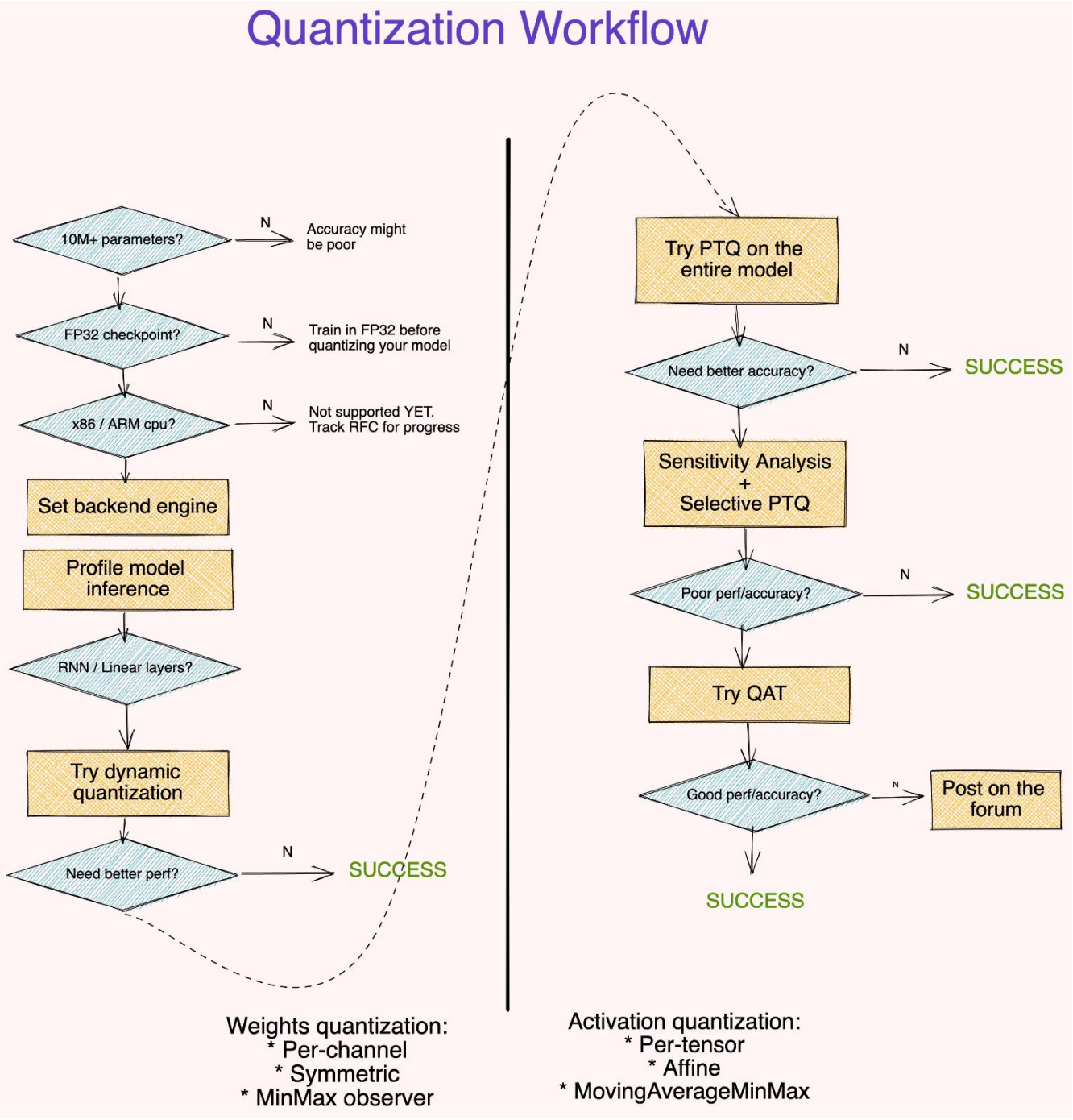


Figure 18: QUANTIZATION WORK FLOW

(https://pytorch.org/blog/quantization-in-practice/#:~:text=FOR%20YOUR%20WORKFLOW-,Fig%209,-.%20Suggested%20quantization%20workflow)

POINTS TO NOTE: **[38]**

- Quantizing a model from a FP32 checkpoint provides better accuracy than training an INT8 model from scratch.
- Use symmetric-per-channel quantization with MinMax observers for quantizing weights. Use affine-per-tensor quantization with MovingAverageMinMax observers for quantizing activations.

6.7. CREATING AN EVALUATION OR INFERENCE GRAPH

Now that we have completed our training and our parameters are now tuned for better low precision inference, we need to obtain a low precision inference graph from the obtained training graph to run it on optimized hardware devices. **[39]**

- First, we need to extract the quantized weights from the above model and apply the Quantize operation to the weights obtained during quant-aware training.
- As our optimized function will be accepting only low precision inputs, we also need to quantize our input.

Now let's derive how we can obtain our quantized result using these quantized parameters.

Suppose we assume convolution as a dot operation.

$$r_3 = r_1.r_2$$

Equation 6

Using Equation 1, It can also be written as:

$$S_3(q_3 - Z_3) = S_1(q_1 - Z_1).S_2(q_2 - Z_2)$$

Equation 7

To obtain the quantized value q_3, we rearrange the equation to be:

$$q_3 = \frac{S_1 S_2}{S_3}(q_1 - Z_1).(q_2 - Z_2) + Z_3$$

Equation 8

In this equation, we can compute $(S_1S_2)/S_3$ offline before the inference even begins and this can be replaced with a single multiplier M.

$$M = \frac{S_1 S_2}{S_3}$$

Equation 9

Now to further reduce it to Integer-only arithmetic, we try to break down M into two integer values. M always lies between 0 and 1, so it can be broken down into this form.

$$M = 2^{-n} M_0$$

Equation 10

Using this equation, we can obtain the integer values of M_0 and n which will act as a multiplier and a bit-wise shifter value respectively. Obviously, this step is not required if we can perform float multiplication on our hardware.

Also, we need to modify our bias accordingly as our multiplier would also be affecting it. Hence, we can obtain our int32 quantized bias for inference using the following equation:

$$b_q = \left\lfloor \frac{b}{S_1 S_2} \right\rceil$$

Equation 11

Now that we have all our ingredients, we can create our low precision inference graph which would look something like this.

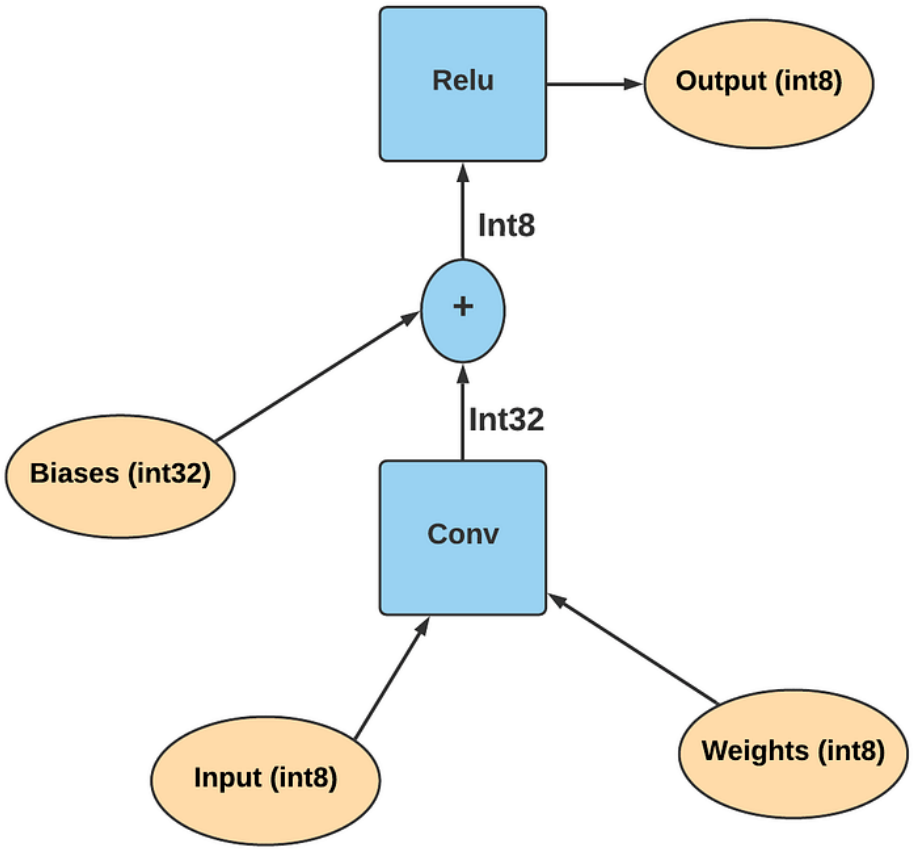


Figure 19: Representation of Quant-Inference Graph

(https://towardsdatascience.com/inside-quantization-aware-training-4f91c8837ead#:~:text=something%20like%20this.-,Figure%203,-.%20Representation%20of%20Quant)

CHAPTER - 7

7. CONCLUSION

In conclusion, this project has offered a comprehensive exploration of Quantization Aware Training (QAT) as applied to YOLOv5, a robust object detection framework. The objective of this work was not merely to conclude the project but to provide a valuable resource for those seeking to leverage the potential of QAT and YOLOv5 within the realm of computer vision. By introducing the concept of QAT and its relevance to YOLOv5, we set the stage for a deeper understanding of how this technique can enhance the deployment of computer vision models on resource-constrained devices.

In the initial chapters, we established the foundation by discussing the principles of computer vision, Convolutional Neural Networks (CNNs), and object detection methods, while our exploration of quantization and its types served as a crucial backdrop for understanding how QAT optimizes YOLOv5 for resource-constrained environments.

Chapter 4 delved into the architecture of YOLO, followed by an examination of PyTorch's role in YOLOv5 in Chapter 5. The core of this project, detailed in Chapter 6, provided a step-by-step breakdown of the QAT process, encompassing key elements such as fake quantization, fakequant, and quantize operations. These operations collectively laid the groundwork for the development of training and quantization workflows, as well as evaluation and inference graphs.

This project equips researchers, developers, and practitioners with the knowledge and insights required to optimize real-time object detection for resource-limited environments. It does not merely conclude the project; rather, it serves as a valuable reference for those aiming to harness the power of QAT and YOLOv5. By providing a comprehensive understanding of the techniques and principles involved, we hope to empower individuals to push the boundaries of computer vision applications, particularly in scenarios where resource constraints are a critical consideration.

7.1. REFERENCES

[1] [1712.05877] Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference (arxiv.org)

[2] [1506.02640] You Only Look Once: Unified, Real-Time Object Detection (arxiv.org)

[3] GitHub - ultralytics/yolov5: YOLOv5 🚀 in PyTorch > ONNX > CoreML > TFLite

[4] YOLO: Real-Time Object Detection (pjreddie.com)

[5] https://en.wikipedia.org/wiki/Computer_vision#cite_note-Klette-2014-1

[6] https://www.ibm.com/topics/computer-vision

[7] https://www.geeksforgeeks.org/introduction-convolution-neural-network/

[8] https://www.geeksforgeeks.org/introduction-convolution-neural-network/

[9] https://www.datacamp.com/blog/yolo-object-detection-explained

[10] https://pjreddie.com/darknet/yolo/

[11] [1512.02325] SSD: Single Shot MultiBox Detector (arxiv.org)

[12] [1311.2524] Rich feature hierarchies for accurate object detection and semantic segmentation (arxiv.org)

[13] https://www.mathworks.com/discovery/quantization.html

[14] https://arxiv.org/abs/1712.05877

[15] https://arxiv.org/abs/1502.02551

[16] https://chat.openai.com/share/ae060881-2626-44bf-bd47-9c64b097157f

[17] https://docs.nvidia.com/deeplearning/tensorrt/tensorflow-quantization-toolkit/docs/docs/qat.html

[18] https://pytorch.org/blog/quantization-in-practice/

[19] https://docs.nvidia.com/deeplearning/tensorrt/tensorflow-quantization-toolkit/docs/docs/qat.html

[20] https://pytorch.org/blog/quantization-in-practice/

[21] https://www.v7labs.com/blog/yolo-object-detection

[22] https://chat.openai.com/share/ea75ec8b-f829-4558-8c4c-892722e3043f

[23] https://www.datacamp.com/blog/yolo-object-detection-explained

[24] https://en.wikipedia.org/wiki/PyTorch#See_also

[25] https://web.archive.org/web/20180615190804/https://pytorch.org/about/

[26] https://pytorch.org/hub/ultralytics_yolov5/

[27] https://chat.openai.com/share/c6343b7a-489c-4985-a2fc-a4c9e965ce16

[28] https://developer.nvidia.com/blog/deploying-yolov5-on-nvidia-jetson-orin-with-cudla-quantization-aware-training-to-inference/

[29] https://chat.openai.com/share/9784892a-6f23-4166-81a2-e9398d6d3c91

[30] https://arxiv.org/pdf/2307.11904.pdf

[31] https://towardsdatascience.com/inside-quantization-aware-training-4f91c8837ead#:~:text=How%20is%20it%20Performed%3F

[32] https://chat.openai.com/c/680a6ca2-a062-4bcd-8a1d-755d286f9ca9

[33] https://towardsdatascience.com/inside-quantization-aware-training-4f91c8837ead#:~:text=something%20known%20as-,FakeQuant,-nodes%20into%20our

[34] https://towardsdatascience.com/inside-quantization-aware-training-4f91c8837ead#:~:text=operations%20stacked%20together.-,Quantize%20Operation,-Its%20main%20role

[35] https://towardsdatascience.com/inside-quantization-aware-training-4f91c8837ead#:~:text=optimize%20the%20model.-,Dequantize%20Operation,-To%20obtain%20back

[36] https://towardsdatascience.com/inside-quantization-aware-training-4f91c8837ead#:~:text=Creating%20a%20Training%20Graph

[37] https://pytorch.org/blog/quantization-in-practice/

[38] https://pytorch.org/blog/quantization-in-practice/#:~:text=for%20larger%20image-,Points%20to%20note,-Large%20(10M%2B%20parameters

[39] https://towardsdatascience.com/inside-quantization-aware-training-4f91c8837ead#:~:text=Creating%20an%20Evaluation%20or%20Inference%20Graph