

**DEEP LEARNING MODEL OPTIMIZATION FOR REAL-TIME SMALL
OBJECT DETECTION ON EMBEDDED GPUS**

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF
ENGINEERING AND NATURAL SCIENCES
OF ISTANBUL MEDIPOL UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF
MASTER OF SCIENCE
IN
ELECTRICAL, ELECTRONICS ENGINEERING AND CYBER SYSTEMS

By

Sharoze Ali

December, 2021

DEEP LEARNING MODEL OPTIMIZATIONS FOR REAL-TIME SMALL OBJECT
DETECTION ON EMBEDDED GPUS

By Sharoze Ali

23 December 2021

We certify that we have read this dissertation and that in our opinion it is fully adequate,
in scope and in quality, as a dissertation for the degree of Master of Science.

Prof. Dr. Hasan Fehmi Ateş (Advisor)

Prof. Dr. Bahadır K. Güntürk

Prof. Dr. Uluğ Bayazıt

Approved by the Graduate School of Engineering and Natural Sciences:

Prof. Dr. Yasemin Yüksel Durmaz

Director of the Graduate School of Engineering and Natural Sciences

I hereby declare that all information in this document has been obtained and presented in accordance with the academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.


Signature :

Name, Surname: SHAROZE ALI

ACKNOWLEDGEMENT

I'd like to convey my heartfelt appreciation to my adviser, Prof. Dr Hasan Fehmi Ateş, for all the help he kindly gave me during the research, as well as for his patience and determination. This effort would not have been possible without his dedication, support, and advice. He taught me a lot and was a great mentor to me. I'm also grateful to my lab colleagues at Istanbul Medipol University for their assistance and fruitful talks. Finally, I'd want to thank my family and loved ones for their encouragement and genuine love in supporting me through the completion of my master's degree and thesis.

I also would like to thank TÜBİTAK for the MS scholarship provided during my graduate study. This thesis is funded by TÜBİTAK-1003 project no: 118E891, titled "Smart Camera System for Wide Area Surveillance".



Sharoze Ali
December, 2021

CONTENTS

	<u>Page</u>
ACKNOWLEDGEMENT	iv
CONTENTS	v
LIST OF FIGURES	vii
LIST OF TABLES	ix
LIST OF SYMBOLS	x
ABBREVIATIONS	xi
ÖZET	xii
ABSTRACT	xiv
1. INTRODUCTION	16
1.1. Literature Review for Multi Object Trackers	18
1.1.1. Joint monocular 3d vehicle detection and tracking	18
1.1.1.1. Datasets	19
1.1.1.2. Experimental setup:	19
1.1.1.3. Results and conclusions	19
1.1.2. Multi object tracking in videos based on lstm and deep reinforcement learning.....	20
1.1.2.1. Datasets	20
1.1.2.2. Experimental setup	21
1.1.2.3. Results and conclusions.....	21
1.1.3. Aerial infrared target tracking in complex background based on combined tracking and detection	22
1.1.3.1. Datasets	22
1.1.3.2. Experimental setup.....	23
1.1.3.3. Results and conclusions	23
1.1.4. Spatially supervised recurrent convolutional neural networks for visual object tracking.....	23
1.1.4.1. Datasets	24
1.1.4.2. Experimental setup:	24
1.1.4.3. Results and conclusions:.....	25
1.1.5. Real-time multi target tracking at 210 mega-pixels/second in wide area motion imagery	25
1.1.6. Real time multi-object-tracking	27
1.1.6.1. Datasets:.....	27
1.1.6.2. Experimental setup	28
1.1.6.3. Results and conclusions	28
2. THEORETICAL PART	29
2.1. Small Object Detection from Aerial Imagery	29
2.1.1. YOLOv4	31
2.1.1.1. Feature aggregation.....	32
2.1.1.2. Spatial pyramid pooling (SPP)	33
2.1.1.3. Freebies	35
2.1.1.4. Specials	35
2.1.2. Improved yolov4 for Aerial Object Detection.....	37
2.1.2.1. Architectural modification	38
2.2. Deep Neural Network Optimization for Real-Time Execution	40
2.2.1. Targeted embedded platform	40
2.2.2. Multi-threading	40

2.2.3.	Tensor RT (TRT)	42
2.2.3.1.	Tensorrt optimizations	43
2.2.3.2.	Layer and tensor fusion:	45
2.2.3.3.	Precision calibration and reduction.....	45
2.2.3.4.	Automatic selection of best kernels	46
2.2.3.5.	Static or dynamic mode	46
2.2.3.6.	Variable batch sizes	47
2.2.3.7.	Regulation of minimum node numbers in tensorrt subgraphs	47
2.2.3.8.	Memory management	48
2.2.3.9.	Quantization-aware training	48
2.2.4.	Deep model compression by parameters reduction	49
2.2.4.1.	Training of base model	50
2.2.4.2.	Sparse training	50
2.2.4.3.	Channel pruning.....	52
2.2.4.4.	Layer pruning.....	53
2.2.4.5.	Network slimming	53
2.2.4.6.	Normal pruning.....	55
2.2.4.7.	Optimized normal pruning.....	56
2.2.4.8.	Shortcut pruning	56
2.2.4.9.	Layer channel pruning	57
2.3.	Multi Object Tracker (MOT) Deployment	57
2.3.1.	The deep sort tracker.....	57
2.3.1.1.	Assignment problem	58
2.3.2.	Joint object detection and embedding (jde) tracker:.....	59
2.3.2.1.	Detection pipeline	60
2.3.2.2.	Appearance embedding.....	60
2.3.3.	Automatic loss balancing.....	61
2.3.4.	Online association.....	61
2.3.5.	Multi-class joint object detection and embedding (mc-jde) tracker.....	61
3.	EXPERIMENTAL PART	63
3.1.	Experimental Setup for Small Object Detection.....	63
3.2.	Experimental Setup for TensorRT Optimizations	66
3.3.	Experimental Setup for Model Compression and Parameters Reduction.....	67
3.4.	Data Augmentation	68
3.4.1.	Experimental setup	68
4.	RESULTS AND DISCUSSIONS.....	70
4.3.	Ablation Study for Pruning Parameters	72
4.4.	Multi Object Tracker (MOT) Deployment	75
5.	CONCLUSIONS AND FUTURE WORK.....	79
	BIBLIOGRAPHY.....	81
	CURRICULUM VITAE.....	87

LIST OF FIGURES

Figure 1.1: Example of Small Object detection from Visdrone-19 DET dataset.....	17
Figure 1.2: Demonstration of Depth Order Matching and Occlusion-aware association.....	19
Figure 1.3: Overview of proposed MOT algorithm.....	20
Figure 1.4: Single Object detector pipeline.....	21
Figure 1.5: CTAD Pseudocode/ working flow.....	22
Figure 1.6: Learning two regression models for a single frame.....	23
Figure 1.7: Overview of ROLO tracking procedure.....	24
Figure 1.8: Propose architecture diagram.....	25
Figure 1.9: WAMI dataset frame and corresponding patches.....	26
Figure 1.10: JDE block diagram.....	27
Figure 1.11: a) JDE architecture diagram, b) JDE network predictions.....	28
Figure 2.1: Inference speed performance comparison, measured in frames per second (FPS) for YOLOv3, YOLOv4 and Faster R-CNN.....	30
Figure 2.2: RCNN Architectural flow diagram.....	31
Figure 2.3: Objector Detector Anatomy.....	31
Figure 2.4: Function network architecture – (a) FPN [52] introduces a top-down direction to fuse multi-scale features from level 3 to level 7 (P3-P7); (b) the PANet [51] adds an extra bottom-up direction to the top of the FPN.....	34
Figure 2.5: Typical spatial pyramid pooling layer in the network where 256 is the filter number of the conv5 layer, and conv5 is the last convolutional layer.....	34
Figure 2.6: SPP block (Separate) and SSP block embedded in YOLO.....	34
Figure 2.7: Mish activation function.....	36
Figure 2.8: An example case (Drop block regularization.....	37
Figure 2.9: Block diagram of YOLOv4 [11] and Modified YOLOv4 [60], where component details in (a) further describes CBM, CBL, SSP [54] etc. blocks used in architectures.....	39
Figure 2.10: Single Threaded pipeline workflow.....	41
Figure 2.11: Multi - Threaded pipeline workflow (Aero head shows the flow of data between 3 independent working threads).....	41
Figure 2.12: Tensor RT Pipeline [1].....	43
Figure 2.13: Vertical fusion Input – Un Optimized Graph [1].....	44
Figure 2.14: Vertical Fusion graph.....	44
Figure 2.15: Vertical + Horizontal Fusion Optimized graph.....	45
Figure 2.16: Block diagram of model conversion pipeline to TRT format.....	47
Figure 2.17: x is the input, r is the tensor floating point range, s is the INT8 scaling factor of the number of values. The above equation takes the x input and returns a quantized value of INT8.....	49
Figure 2.18: The effect after sparsifying the given tensor, resulting 2 times faster execution.....	51
Figure 2.19: Accuracy restoration while sparse training.....	52
Figure 2.20: Network Slimming flow chart.....	54
Figure 2.21: mAP VS FPS plots of object detection models; FP16 and FP32 represents corresponding TensorRT models. FP-16 models are more directed towards real-time.....	59
Figure 2.22: Plot shows accuracy improvement after including Augmented data (Tiles + Zooms) in Visdrone DET dataset.....	62
Figure 3.1: mAP results of original and modified YOLOv4 [60] for all object classes at 832x832 image resolution.....	64

Figure 3.2: mAP calculated on original and modified YOLOv4 for different image resolutions at test stage [60].....	65
Figure 3.3: FPS results of original and modified YOLOv4 [60] for different image resolutions at test stage	65
Figure 3.4: Plot shows accuracy improvement after including Augmented data (Tiles + Zooms) in Visdrone DET dataset.	69
Figure 4.1: Comparison of original and modified YOLOv4 in terms of mAP and FPS for different image resolutions on Jetson Xavier [60].	71
Figure 4.2: mAP VS FPS plots of object detection models; FP16 and FP32 represents corresponding TensorRT models. FP-16 models are more directed towards real-time.	72
Figure 4.3: Plotting trainable parameters in millions (Red color) reduced after applying pruning techniques and corresponding highest FPS (Orange color) as compared to base models.	75
Figure 4.4: Results evaluated on VisDrone-MOT2021 Challenge's (MOT) data. Each assessment mode's top three findings are bolded and highlighted. Red, green, and blue are used as accent colors.[74].....	76

LIST OF TABLES

Table 2.1: Table FPS Comparison among Single-Threaded VS Multithreaded pipeline on RTX 2080TI.....	42
Table 2.2: FPS Comparison among Single-Threaded VS Multithreaded Pipeline on AGX Xavier. Here FPS didn't increase due to a smaller number of CPUs cores in Jetson AGX Xavier (8 CPU Cores) as compared to previous table for RTX 2080TI Server (32 CPU Cores).	42
Table 3.1: T Table contain hardware specifications of machine being used for experiment	63
Table 3.2: mAP VS FPS of original (YOLOV3, YOLOV4 and Improved YOLOv4) Original models before serializing to TRT format evaluated on Visdrone Test-Dev Dataset on Jetson AGX Xavier.....	66
Table 3.3: mAP VS FPS on Visdrone Test-Dev Dataset with TensorRT FP32 models on Jetson AGX Xavier.....	66
Table 3.4: mAP VS FPS on Visdrone Test-Dev Dataset with TensorRT with FP16 on Jetson AGX Xavier.....	67
Table 3.5: Table shows the results against applied pruning techniques on YOLOv3 detection network. A higher reduction in model size and number of network layers and Increase in FPS and mAP indicates a better mode.....	68
Table 4.1: FPS of pruned model (Normal Pruning) on prune ratio 40. More than 40% mAP reduce to zero.....	73
Table 4.2: . FPS of pruned model (Slim Pruning) on different prune ratios.....	74
Table 4.3: Table FPS Comparison among Single-Threaded VS Multithreaded pipeline on RTX 2080TI.....	74
Table 4.4: T FPS of pruned model (Layer Shortcut Pruning) on different prune ratios.....	75
Table 4.5: FPS and accuracy comparison among Deep Sort and MC-JDE tracker obtained from VisDrone 2018 MOT toolkit evaluated on VisDrone MOT Test-dev dataset.....	77
Table 4.6: Comparison among other tracker on VisDrone MOT Test-Dev data. (Note: The Deep Sort and MC-JDE tracker are not trained on MOT dataset). Bold figures shows better results of our trackers without training on MOT dataI.....	78

LIST OF SYMBOLS

α	: Learning Rate
γ	: Channel Cutting Threshold
β	: Channel
$>$: Is Greater Than
λ	: Scaling Factor
\approx	: Approximately Equal



ABBREVIATIONS

WAS	: Wide Area Surveillance
MOT	: Multi Object Tracking
CNN	: Convolutional Neural Network
UAV	: Unmanned Aerial Vehicles
CPU	: Central Processing Unit
GPU	: Graphics Processing Unit
LSTM	: Long Short-Term Memory
SVM	: Support Vector Machine
RNN	: Recurrent Neural Network
SAT	: Self Adversarial Training
CTAD	: Combine Tracking and Detection
VGG	: Visual Geometry Group
JDE	: Joint Detection and Embeddings
WAMI	: Wide Area Motion Imagery
YOLO	: You Only Look Only Once
COCO	: Common Object in Context
R-CNN	: Region Based Convolutional Neural Network
CSP	: Cross Stage Partial Network
CMBN	: Cross Mini-Batch Normalization
CBL	: Convolution - Batch Normalization - Leaky Relu
SPP	: Spatial Pyramid Pooling
PAN	: Path Aggregation Network
FPN	: Feature Pyramid Network
FPS	: Frames Per Second
MAP	: Mean Average Precision
FP	: Fixed Precision
AP	: Average Precision
IOU	: Intersection Over Union
NMS	: Non-Max Suppression
RAM	: Random Access Memory

GÖMÜLÜ GPU'LARDA GERÇEK ZAMANLI KÜÇÜK NESNE TESPİTİ İÇİN DERİN ÖĞRENME MODEL OPTİMİZASYONU

ÖZET

Sharoze Ali

Elektrik-Elektronik Mühendisliği, Yüksek Lisans

Tez Danışmanı: Prof. Dr. Hasan Fehmi Ateş

Aralık, 2021

Kameralı hava araçları genellikle gözetleme alanında kullanılmaktadır. Bu gözetleme sistemlerinin birçoğu objeleri iki adımda izler; ilk olarak, hedeflerini tespit eder ve tanır, daha sonra bu hedefleri canlı video akışında takip eder. Fakat, günümüzde obje tespit algoritmaları genellikle yüksek hesap gücü olan ve GPU ile desteklenen sistemlerde, geniş görüntü veri seti ile eğitilen derin öğrenme modelleri kullanılmaktadır. Dahası, obje takibinde öznitelik eşleştirme ve ilişkilendirme yönetimleri sisteme daha fazla iş yükü bindirmekte ve bu gerçek zaman performansını etkilemektedir.

Geniş Alan Gözetleme (GAG) uygulamalarında, yer hedeflerini bulmak ve takip etmek için görüş tabanlı nesne algılama ve hedef takibi gereklidir. Fakat, bu insansız hava uçakları (İHA) yerden çok yüksekte çalışır ve bu sebeple yerdeki objeler çok küçük görünür. Bu sebeple, özellikleri derinlemesine tarayabilen ve bu küçük yer nesnelerini tanıyabilen hassas bir obje dedektörü gereklidir. Ayrıca, derin öğrenme yaklaşımını kullanan CNN tabanlı obje dedektörleri, kapsamlı hesaplamaları ve karmaşık matematiksel modelleri nedeniyle gömülü veya uç cihazlarda çalışmak için ağırdır.

Bu tezde, Çoklu Nesne Takibi (ÇNT) sisteminin performansını etkileyebilecek yukarıda belirtilen sorunları araştırdık. Bu tezin amacı, Nvidia Jetson AGX Xavier gibi uç gömülü cihazlarda etkin bir şekilde çalışabilen, gürbüz ve gerçek zamanlı bir takip sistemi tasarlamaktır. İlk olarak, tek aşamalı bir dedektör seçtik ve küçük objeler için daha rafine ve ince tanecikli öznitelikler elde etmek adına yukarı örnekleme katmanlarını bağlayarak ve yukarı örnekleme öznitelikleri orijinal özniteliklerle birleştirerek mimari tabanlı öznitelik iyileştirme üzerinde çalıştık. Bu da daha kesin obje algılamaya yardımcı oldu. Daha sonra, karmaşık obje tespit modellerini hafif sistemlere dönüştürmekle ilgili, olası (CPU ve GPU tabanlı) model optimizasyon yaklaşımlarını keşfettik. Bu yaklaşımlar, TensorRT [1] boru hattı kullanılarak yapılan karışık hassasiyet optimizasyonu ve katman füzyonu, çok-izleklili programlama ve çeşitli budama tekniklerini içermektedir. Bu tekniklerin kullanılması obje tespit modelimizin doğruluk ve verimlilik hedeflerinden ödün vermeden gerçek zamanlı sonuçlar almamızı sağlamıştır. Ayrıca, Visdrone [2] tespit veri seti üzerinde veri artırma teknikleri uyguladık ve bu da test veri setinde daha yüksek ortalama kesinliğe yol açtı. Benzetim sonuçları, farklı tespit/takip modelleri ile bunların optimize edilmiş sürümleri arasındaki performans açısından kapsamlı karşılaştırmaları göstermektedir.

Anahtar sözcükler: Derin Öğrenme, Nesne Tespiti, Nesne Takibi, Gözetim, Gezinge, Çok-izleklî Kodlama, Karma Hassasiyet, Veri Artırma, Gömülü Platform, Seyrek Eğitim, Ağ İnceltme.



DEEP LEARNING MODEL OPTIMIZATION FOR REAL-TIME SMALL OBJECT DETECTION ON EMBEDDED GPUS

ABSTRACT

Sharoze Ali

MSc in Electrical, Electronics Engineering and Cyber Systems

Advisor: Prof. Dr. Hasan Fehmi Ateş

December, 2021

Camera mounted drones are mostly used in surveillance applications. Most of these surveillance systems track objects in two steps; firstly, they detect and recognize targets in a scene and then track those targets in the upcoming live video feed. However, current object detection algorithms mostly use deep learning models that are trained on large image datasets that require high computing power and GPU supported systems. Moreover, feature matching and association handling in object tracking also create more payload on a system that affects performance in real-time.

For Wide Area Surveillance (WAS) applications vision-based object detection and target tracking is necessary to locate and follow ground targets. However, these UAVs operate at very high altitude above the ground, due to which ground objects look very small and less visible. Hence an accurate object detector is needed which can deeply scan the features and recognize these small ground objects respectively. Meanwhile, using the deep learning approach, CNN based object detectors are heavy to operate on embedded or edge devices because of their extensive computation and complex mathematical models.

In this thesis we investigate both above stated problems that can affect the performance of Multi Object Tracking (MOT) system. The motivation of this thesis is to design a robust and real-time tracking system that can operate effectively on edge embedded devices like Nvidia Jetson AGX Xavier. First we choose one-stage detectors and work on architectural based feature enhancement by connecting Up-sampling layers and concatenating the up-sampled features with the original features to obtain more refined and grained features for small objects, which leads to more accurate small object detection, and Second we explore possible (CPU and GPU based) model optimization approaches involved in transforming the complex object detection models to lightweight systems, which include mix precision and layers fusion using TensorRT [1] pipeline, multi-threading, and several pruning techniques to work our object detection models for real-time performance, without sacrificing accuracy and efficiency goals. Moreover, we apply data augmentation techniques on Visdrone detection dataset [2] which also lead to improved mean Average Precision (mAP) [3] on the test dataset. Simulation results show extensive comparisons in performance between different detection/tracking models and their optimized versions.

Keywords: Deep Learning, Object Detection, Object Tracking, Surveillance, Tracklets, Multi-Threading, Mix Precision, Data Augmentation, Embedded Platform, Sparse Training and Network Slimming.



CHAPTER 1

1. INTRODUCTION

Unlike traditional approaches, deep learning has provided highly accurate results and is verified method in Machine Vision field. Hence state-of-the-art object recognition methods like SSD [4], Faster-RCNN [5][15], Mask-RCNN [6], and YOLO [7] all are based on deep learning architectures, which are involved in several applications such as surveillance, robotics, and other dominant fields like natural language processing (NLP) and drug discovery. In this thesis we discuss different deep learning based Multiple Object Tracking approaches and try to figure out the issues in way towards robust real-time solutions.

In recent literature, Deep Learning (DL) approach is widely used in Object detection, Tracking by Detection and online Multiple Object Tracking (MOT), and this is because Convolutional neural Networks (CNNs) have ability to extract deep spatial patterns from input data. Thus, their ability of learning complex and rich features from their input results in accurate and reliable output. There are also other forms of architectures like recurrent neural networks (RNNs) [8] and Long Short-Term Memory (LSTM) [9] used to process consecutive video frames to track objects with their IDs. Multiple Object Tracking (MOT) process is further divided into the following sub problems:

- Acquiring targets using robust detection algorithm.
- Simultaneous focusing on all the tracklets while handling partial occlusions.
- Object deformation.
- Motion blur.
- Illumination and scale variations.

In literature review we will discuss the list of different applied algorithms that help us in understanding; how deep learning works in Multiple Object tracking either by the online simultaneous tracking or with tracking by detection approach.

Accurate detection of small objects is another well-known issue in drone surveillance systems. Conventional methods rely on hand-crafted image characteristics and background reduction methods. However, these techniques suffer from a lack of robustness in the context of variations in lighting, weather, position, and so on. Deep learning-based object detection models were developed with the launch of Convolutional Neural Networks. The YOLO [7][10][11] family is made up of single stage object detection models, which we have largely used in our investigations on UAV images.

Meanwhile, simultaneous Visual Object detection and tracking is also vastly studied topic in Computer-Vision and Robotics. Most of the surveillance systems work on this technique in which the system firstly recognizes objects in a scene and then tracks the target in frames of live video feed. However, object detection mainly uses a deep learning approach by training on a large image dataset that requires high computing power and GPU supported systems. Moreover, object tracking in parallel creates more payload that affects performance in real-time. To run these surveillance applications on UAVs having compact on-board processors, it is necessary to design robust and real-time systems. In this work, we discuss an in-depth comparison of Object detection and tracking algorithms and figure out the most applicable ones that are accurate as well as real-time. Later we also show modifications for optimization of these algorithms and network models.

Figure 1.1 shows small object detection in UAV captured outdoor imagery.



Figure 1.1: Example of Small Object detection from Visdrone-19 DET dataset [2].

1.1. Literature Review for Multi Object Trackers

Literature review includes detailed discussion of different object detectors as well as multi object trackers. This chapter discusses the list of different applied algorithms that help us in understanding how deep learning works in Multiple Object tracking either by joint detection and tracking solution or with tracking by detection approach.

For the most part, tracking by detection approach is used to differentiate between all tracklets. This approach mainly works sequentially; A video frame is fed into the system by external camera or webcam, and further proceeded by deep neural nets like (CNN) that finally detect and classify the objects present in the video frames by localizing them with bounding boxes. Here, tracker is initialized, which takes Region of Interests (ROIs) of each detected object and passes them to another network to create embeddings to differentiate between each detected object present in the scene, where ROIs are the bounding box locations returned by object detector.

The second approach works by taking input data in the form of video frames from live camera feed. The main difference is that the same detection network that recognizes objects is also used to generate corresponding embeddings as well. Some MOT datasets, like Visdrone, provide additional information, like learning partial occlusions, custom occlusion ratio (1% ~ 50%), heavy occlusions (50% ~ 100%), which informs targets covering with each other and truncation ratio which indicates the degree of object parts appearing outside a frame. Such parameters are also learned additionally with detection losses and embedding learning tasks.

1.1.1. Joint monocular 3d vehicle detection and tracking

This strategy [12] claims a novel online framework for 3D vehicle detection and tracking on monocular video frames. It detects 2D as well as 3D bounding box information from 2D images. Moreover, for robust instance associations, the depth of 3D bounding boxes is calculated, for handling occlusions, it predicts 3D trajectory for re-identification of occluded vehicles and finally, for long-term and accurate motion extrapolation, LSTM motion learning module is also designed.

1.1.1.1. Datasets

All simulations for the 3D tracking pipeline are done on KITTI [13] and Argoverse [14] dataset. Besides these two, the authors also report some of the game datasets like FSV, GTA5, Cityscapes in their manuscript.

1.1.1.2. Experimental setup

Phrasing 3D tracking as a supervised learning problem, it was the aim to find N trajectories against N detected objects in a video, and these trajectories link a sequence of detected states starting from first visible frame ' a ' to the last visible frame ' b '. In the paper, authors employed Faster R-CNN [5] trained on their custom dataset to grab object proposals as bounding boxes. These boxes correspond as 2D as well as an estimated projection as 3D bounding box center which is further used in prediction of the whole 3D box with other necessary function. The loss function used while predicting is L1 loss. Finally, robust linking across frames in addition to occlusion aware association and depth-order matching is produced by LSTM [9] based tracker.

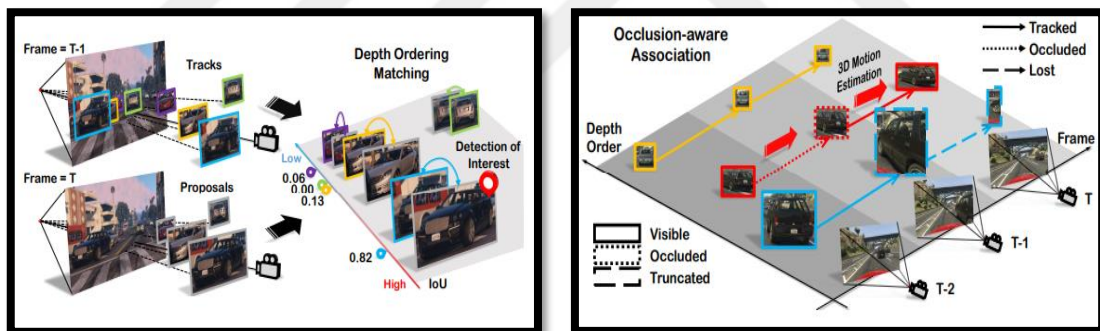


Figure 1.2: Demonstration of Depth Order Matching and Occlusion-aware association. [12].

Training procedure involves 4 GPUs working parallel, with batch size 20 and resolution up to 1920 x 1080 for GTA. 1920 x 1216 for Argoverse and 1248 x 1216 for KITTI dataset [13] train on Faster-RCNN RPN. Tracking policy involves keeping all tractlets until they disappear from certain range (10m to 100m) distance from camera.

1.1.1.3. Results and conclusions

Current work provides the problem of detection and tracking with sub problems, their in-depth scanning and organized solution, however the system requires numerous high-

power computing devices to run in production. For embedded on-board units, a lightweight solution is required.

1.1.2. Multi object tracking in videos based on LSTM and deep reinforcement learning

This study reports on the tracking of several objects on video using LSTMs and deep reinforcement learning. They employ the YOLOv2 architecture for detection, and the Single Object Tracking problem as a Markov decision process for tracking [16]. The network configuration of the single object tracker includes a CNN unit followed by an LSTM unit. Deep reinforcement learning agents are used to train each tracker. Here LSTMs are used in conducting data associations for each frame among result returned from object detector and single object tracker.

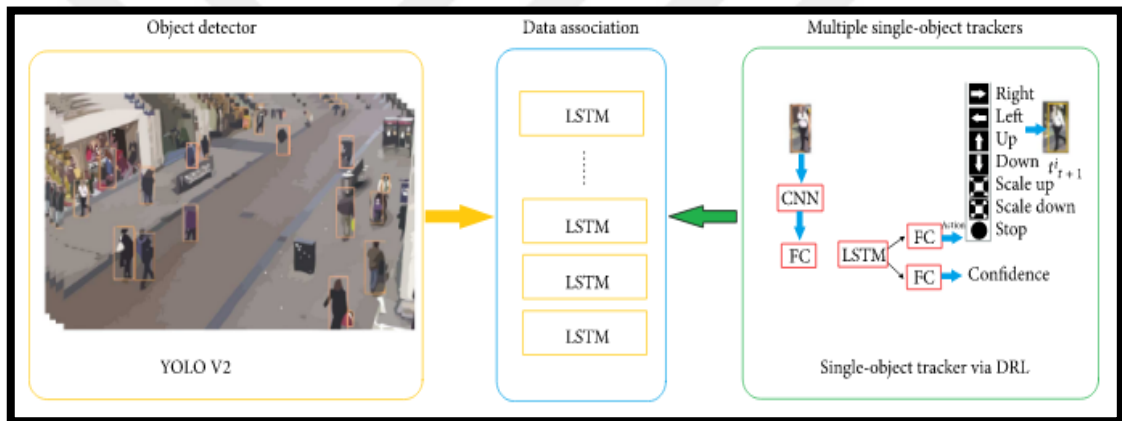


Figure 1.3: Overview of proposed MOT using LSTM and deep reinforcement learning [17].

1.1.2.1. Datasets

The system is trained on pedestrian datasets which contains mixture of other pedestrian benchmarks like PETS09-S2L2 [18] consists of 436 samples with (768 * 576) pixels with handling several occlusions and scale changes. Other dataset sequences like ADL-Rundle-3 consists of 625 samples of (1920 * 1080) dimensions, AVG-Town Centre and TUD [19] which consists of side view of people crossing a road with 201 samples and (649 * 480) frame size also been considered for performance evaluation.

1.1.2.2. Experimental setup

As previously stated, the YOLOv2 [7] architecture was utilized to detect items passed on each frame. Second, the single object tracker is made up of LSTM, which marks the SOT as a Markov decision problem (MDP) by making a series of decisions. MDP provides a set of actions that are applied to the bounding box and one action that finishes the process. Action is stored as a seven-dimensional vector and is divided into three subsets: horizontal moves right and left, vertical travels up and down, and scale changes scale up and scale down.

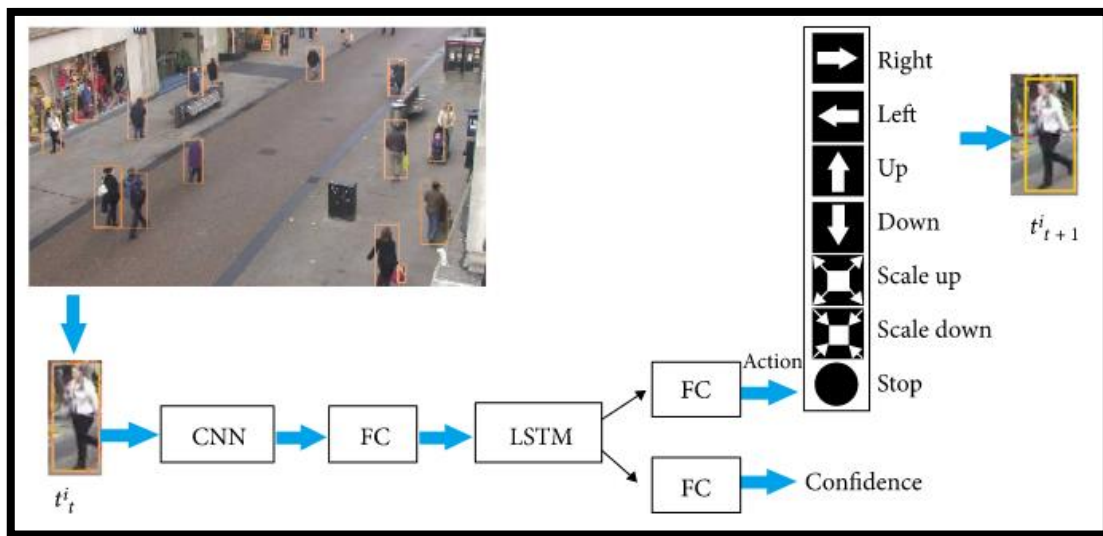


Figure 1.4 Proposed Single Object detector pipeline with LSTM state predictor [17].

Deep CNN consist of VGG-16 [20] network. The data association module based on LSTM [8] has two layers and 512 hidden units, which trains SOT in 40 hours. The current work is MATLAB [21] based and simulated on a system having Core i7 CPU and Nvidia Titan X GPU with 12 GB RAM.

1.1.2.3. Results and conclusions

The existing system can be made real-time replacing other light detection network or slimming the architecture and then retrained it again. To implement we must reproduce algorithm and rewrite the code as all experimental work and dataset was not provided, and the system was built on MATLAB [21].

1.1.3. Aerial infrared target tracking in complex background based on combined tracking and detection

Aerial infrared target tracking is normally used in weapon system that especially the air-to-air missile. This paper focus on developing an algorithm that could track the aircraft fast and accurately based on infrared image dataset. This framework propose tracker based on correlation filter and deep learning-based detector, which it mentions as combine tracking and detection (CTAD) [22]. It claims high efficiency provided by correlation filter and can track the infrared targets reliably. One mentioned in their work that beside there CNNs are highly feature extractors but it's not able to use them in real-time as computationally, it is even more then whole working pipeline of their target tracking bird. Hence, they choose CTAD which use two regression models based on correlation tracking. They choose LCT tracker [23] as a base which is common for long time tracking, further based on DSST [24], so they could fit the need of Infrared (IR) image guidance missile. Tracking performance is continuously maintained by checking the confidence score, so when it drops the certain limit, again the target is detected and vice versa.

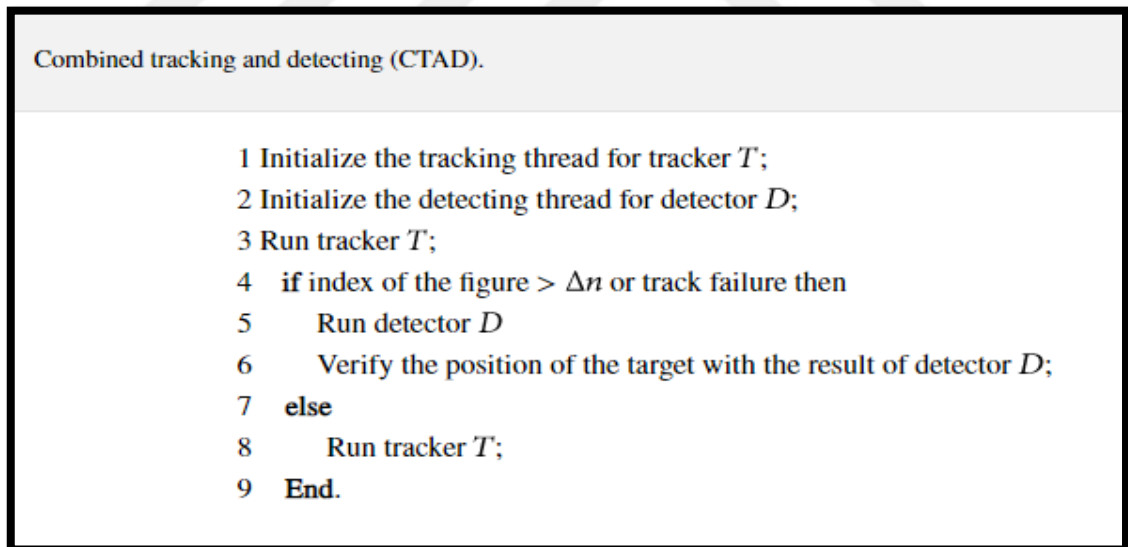


Figure 1.5: CTAD Pseudocode demonstrating CTAD tracker workflow in steps [22].

1.1.3.1. Datasets

All experiments are performed on three different infrared image sequences from VOT-TIR2016 [25] datasets, however for evaluation two sequences are taken from AMCOM FLIR [26] dataset.

1.1.3.2. Experimental setup:

This work pattern is again very similar to previous approaches as discussed earlier like using CNN feature extractors as detection but for tracking they used CTAD [22] that out-performed other with comparable speed of about 18.1 fps. They used correlation filters approach which measures the similarity among two different signals. The algorithm contains two main parts, first CTAD tracker and YOLOv3 [10] as verification for detector. This tracker is implemented in MATLAB with TensorFlow [27] framework on Nvidia GTX1080 GPU with 4GB memory. Trained data consist of 2200 IR images with 5000 epochs on YOLOv3 [10].

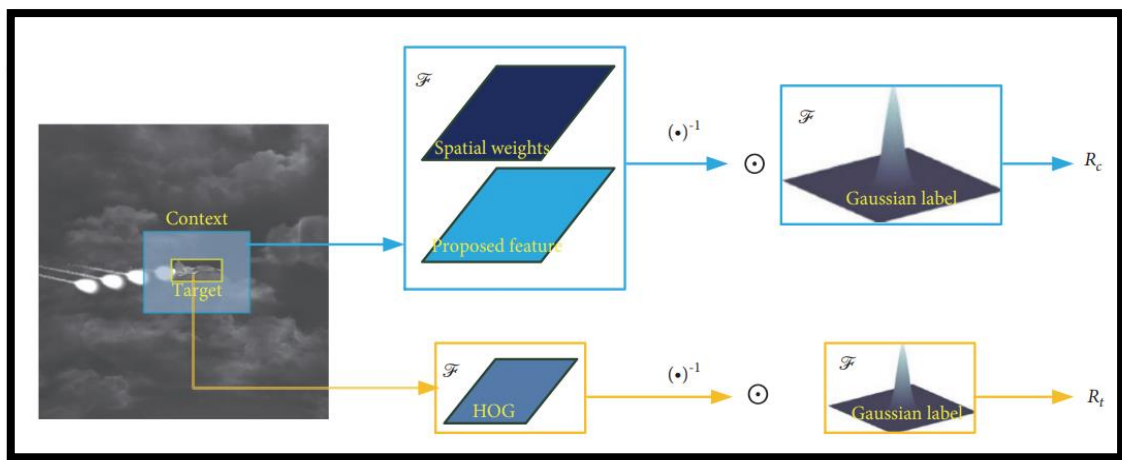


Figure 1.6: Learning two regression models for a single frame to predict future position in next frame [22].

1.1.3.3. Results and conclusions

To the best of our knowledge YOLOv3 is more sensitive in detection for small objects, as it uses multi scale feature fusion, and predict bounding boxes at different scales and fast and accurate. CTAD claims superior performance for real-time scenarios but in short, its single object tracker and implemented on MATLAB [21] so it conflicts to ours approach in data, which is IR data, and non-provided opensource help.

1.1.4. Spatially supervised recurrent convolutional neural networks for visual object tracking

This strategy [28] consists of recurrent convolutional network that utilize tractlets location history with visual features learned by deep neural net. The work also discusses regression capabilities of LSTMs (Long Short-Term Memory) in temporal domain and

tried to concatenate high level visual features from CNN with region information to keep track on obtained targets. Unlike other deep learning-based trackers, this approach use regression for prediction of tracking locations at both conv layer and recurrent unit which results in accuracy as well as robustness as compared to other tracking systems. Choosing YOLOv2 [7] at first stage for collecting visual features and location inference as B-Box, they also use LSTM in the 2nd stage which is spatially deep and appropriate for sequence processing. They named this approach as ROLO: Recurrent-YOLO [29]

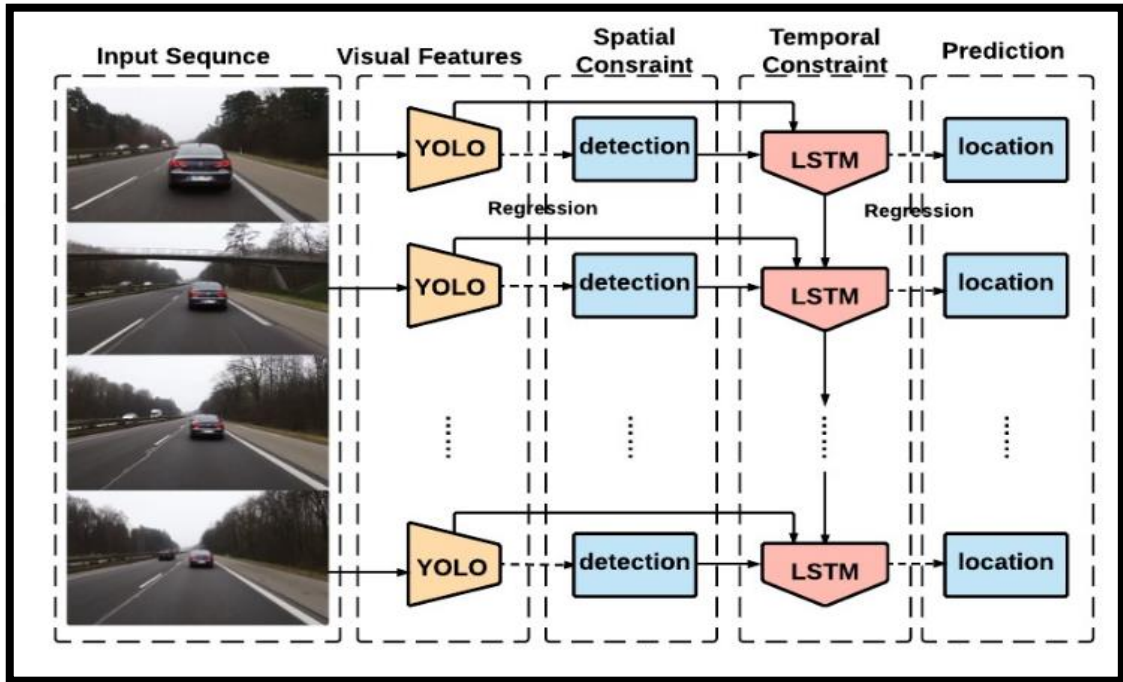


Figure 1.7: Overview of ROLO tracking procedure: YOLO detection + LSTM [29].

1.1.4.1. Datasets

The convolutional weights are learned with ImageNet [30] data of 1000 classes such that network can understand arbitrarily objects in general. While training, the feature vector is fed into SVM [31] (Support Vector Machine) classifier to get good classification results. Further the model is also fine-tuned at PASCAL VOC [32] dataset.

1.1.4.2. Experimental setup

While training network at detection stage, a mid-level resolution feature vector of size 4096 is fed into classifier. Once the network is trained and able to generate visual

features, YOLOv2 architecture is adopted as detection module. At last LSTM-RNNs is added for the training of tracking module. LSTMs is taking two stream data flow into it, such as feature representation and detection information $(0, x, y, w, h, 0)$. In addition, another input is also taken by LSTM [8].

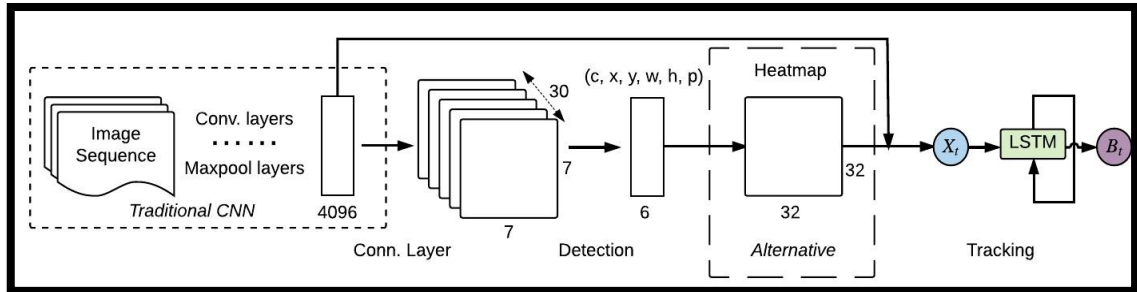


Figure 1.8. Proposed architecture flow diagram of ROLO MOT [28].

which is the output of states from the last time-step. Mean Square Error is used for training purpose. Current system is implemented in Python on TensorFlow framework [27] with 8 cores CPU and NVIDIA TITAN-X GPU system. The work is opensource and is available on GitHub.

1.1.4.3. Results and conclusions

Using LSTMs with object detector solution was already sort out in my future work. Meanwhile while studying the case, we prepared a power-point presentation and implement the solution on the CPU base system. I observed that occlusions on the validation dataset were handled very finely. On CPU the system was not looking much robust, but again to test, retrained, or fine-tune the whole architecture computational resources are required. Taking one further step, one developer tried to convert the whole system on multiple objects tracking and named as MOLO (Multi Object-YOLO), but it is still under development in solving issues.

1.1.5. Real-time multi target tracking at 210 mega-pixels/second in wide area motion imagery

Though this working approach [33] was old, but we studied it as their environment resembles with our working environment. The system takes the big size aerial images which can cover about (6 – 50) km of the total area and many small objects moving in the scene., so the task is to track all these objects.

The dataset they used is WAMI [34][35] (Wide Area Motion Imagery), from Airforce Research Lab (AFRL's) also called AFRL/WPAFB 2009 datasets [35]. The huge size images from this dataset are not such an easy to process and apply tracking algorithm. To solve the problem, the whole frame is divided into small tiles and are maintained by postgre SQL database [36]. Each tile is processed independently and are linked finally with inter-tile linker module.

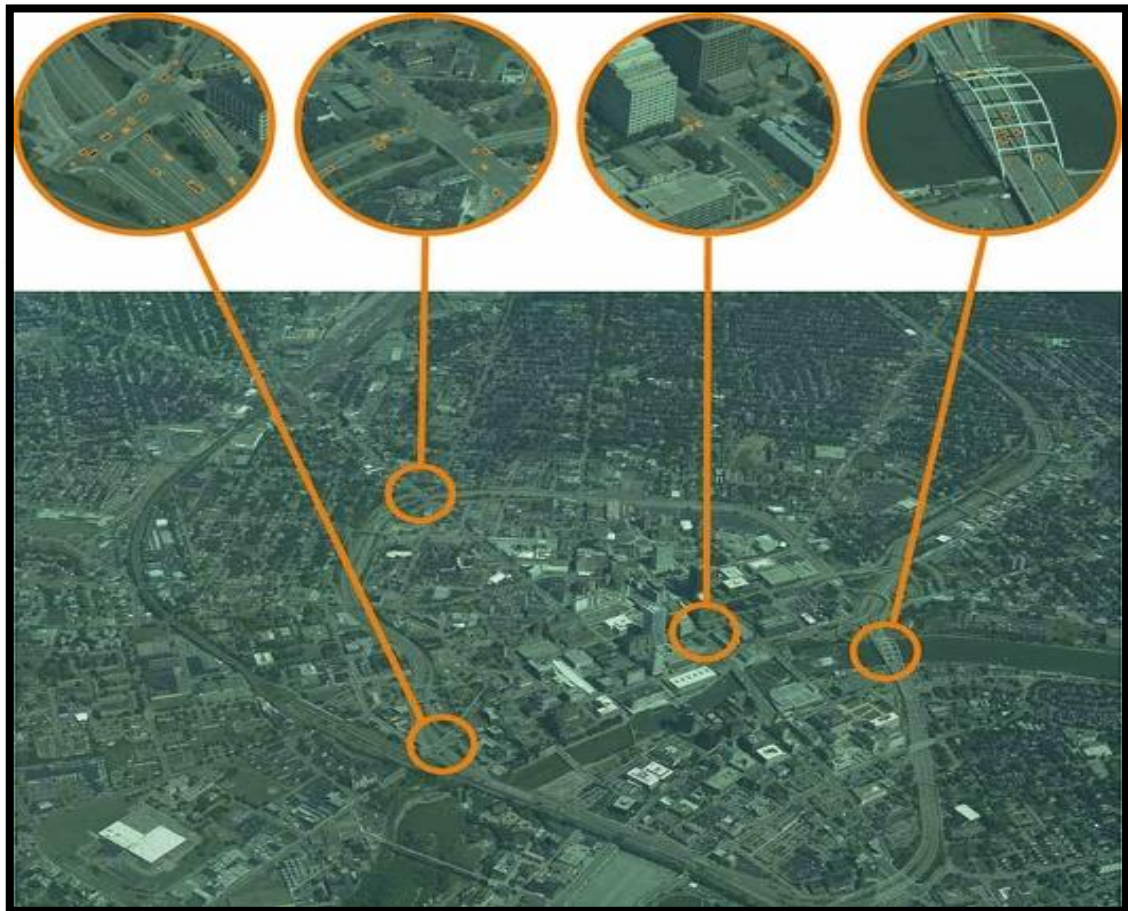


Figure 1.9: WAMI dataset frame and corresponding patches [33].

According to the claims, the suggested system has a real-time data throughput of at least 210 megapixels per second. This tracking system is multi-threaded based and operates by decoding pixels, stabilizing the image with frame differencing, and detecting motion with frame-to-frame homographies. On each frame, they take as assignment problem by Hungarian algorithm to optimally assign new detections to existing tracks. The complexity of the algorithm is $O(n)^3$ and runs on the cluster on which 12 super Micro blades are configured with high-performance database fast disk array and InfiniBand high-speed network for fast access of images. Beside these reviews there were other manuscript which we go through and mark them in secondary category.

- KWIVER framework [37] (Video exploitation application for wide area surveillance)
- KTH Master's thesis: Tracking humans in video using LSTM Recurrent Neural Network [38].
- Cross Input Neighborhood Difference for Re-identification of humans [39].
- Deep Sort [40].

1.1.6. Real time multi-object-tracking

Being focused to make real-time solution, this work adopts tracking-by-detection method and proposes real-time MOT system that enables for the learning of target detection and appearance embedding in a shared model; hence called Joint Detection Embeddings [41] (JDE). This detection model not only return detection but also embeddings associated with each detected object. Thus, this approach is better than the other tracking by detection models in which detection model another embedding model work separately to generate detections and then embeddings respectively, to differentiate to generate final tracklets.

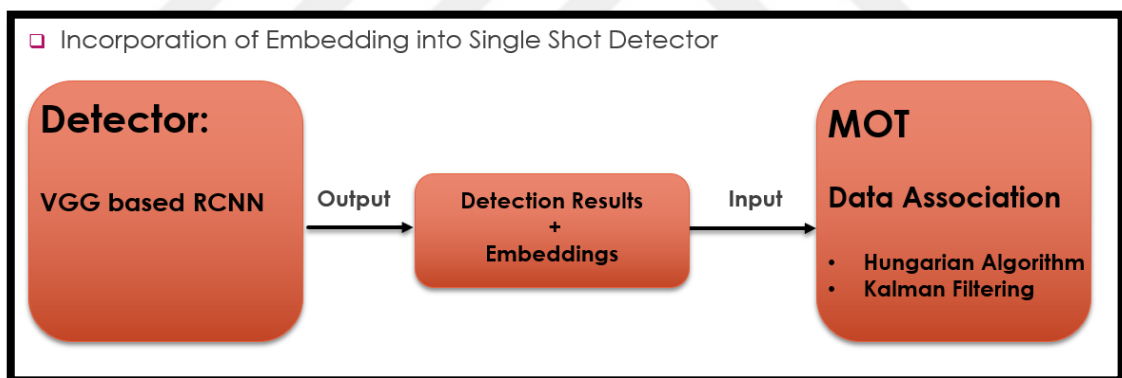


Figure 1.10: JDE block diagram with sub modules.

Normally, process of detection and tracking and their associations results in efficiency problems so the feasible idea of joining detecting and embedding model into a single network might reduce the computations because here these two tasks share the same set of low-level features.

1.1.6.1. Datasets

This work has been done after collecting, six different datasets on pedestrian detection and person search and then refined labelled with bounding boxes and portion of their

identities. Their results performance compares with other known benchmarks like MOTA and MOT-16.

1.1.6.2. Experimental setup:

The current work approaches Feature Pyramid Network [42] (FPN) as its base architecture which makes prediction from multiple scales, which is a general approach to gain accuracy especially when target scale varies a lot. For loss calculation, embedding learning, triplet loss is used. Finally, in addition, the work also introduced online association strategy to work with JDE [41] in which affinity matrix among all observed and previous embeddings are calculated, then with the help of Kalman filtering [45] the new places of tractlets are predicted. Hence if predicted location is too far then assignment is rejected and tractlets are updated.

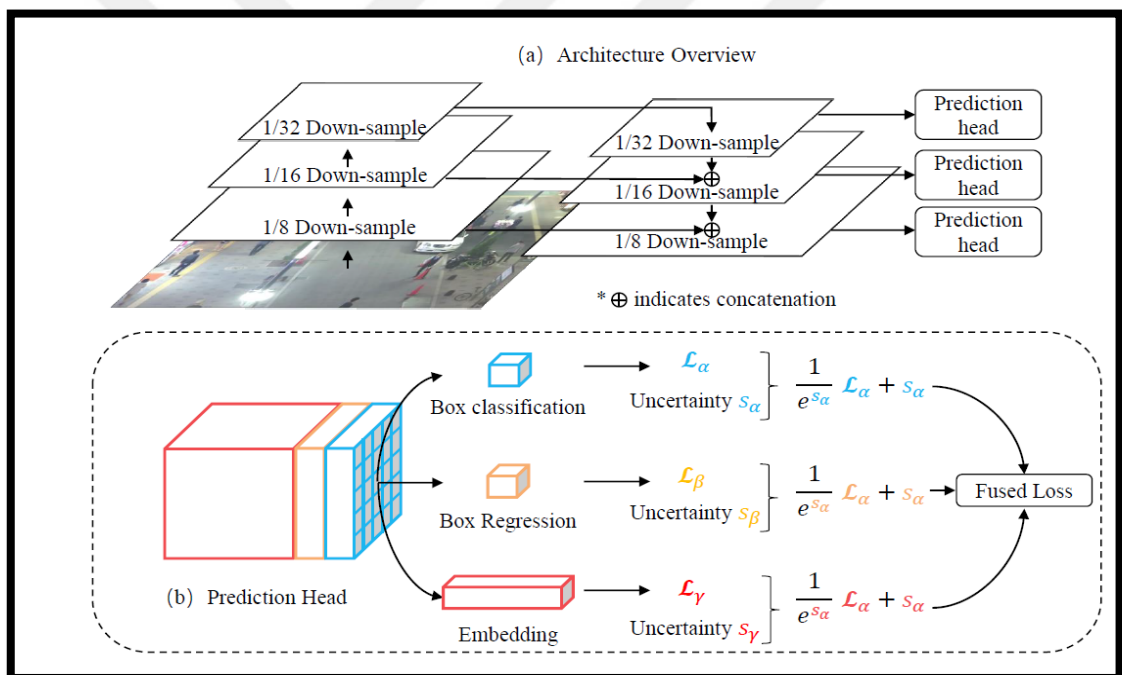


Figure 1.11: a) JDE architecture diagram, b) JDE network predictions [41].

1.1.6.3. Results and conclusions

On comparison, this work performed well compared with its predecessor MOT algorithms. This approach claims (18 to 24) FPS on Nvidia Titan X. To use it for embedded platform, one must put more efforts to make it more computationally effective.

CHAPTER 2

2. THEORETICAL PART

2.1. Small Object Detection from Aerial Imagery

Most surveillance systems normally follow two steps to operate: first, they identify and recognize targets in a scene, and then they track those targets in subsequent video frames. However, deep learning-based object detection needs high performance GPU-enabled systems. On the other hand, feature matching and association in object tracking also add more payload on a system, affecting real-time performance. So simultaneous target identification and tracking on the UAV's onboard processors requires a real-time tracking system that can operate effectively on edge embedded devices like the Nvidia Jetson Xavier.

We divide this problem into two subparts to improve the performance of the real-time embedded platform: the detector part and the tracking part. From the literature review, Joint Detection and Embedding (JDE) [41] approach stand better from others in which the same deep network is used for detection as well as to generate embeddings accordingly. This increases FPS, and it is suitable for real-time operation.

We review the performance of several state-of-the-art object detectors, bottle necks and their execution times. However, we only consider the task of small object detection because in our case video feed is taken from cameras mounted on drones, so object size relative to image size is very low in frames captured by drone cameras. Due to high altitude of drones, objects on the ground look small. Recently one research article is published which compare the performance of existing object detectors on vehicle datasets based on aerial images. This comparative study helps us in choosing best detector according to our problem sets. **Figure 2.1** shows performance comparison of latest object detection algorithms.

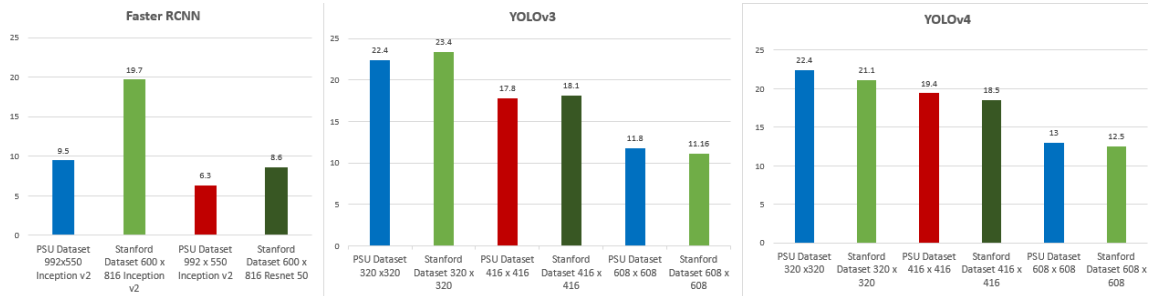


Figure 2.1: Inference speed performance comparison, measured in frames per second (FPS) for YOLOv3, YOLOv4 and Faster R-CNN. [46].

In chapter 2, we have seen that the detector used in the pipeline of Joint Detection and Embedding (JDE) [41] is Faster R-CNN [5] which is the updated version of R-CNN [43] detector. RCNN is a two-stage detector. It first extracts a predefined number of areas (i.e., 2000 region proposals) via a selective search and then applies a greedy approach to combine comparable regions to provide candidate regions for object detection. Later another version named Faster R-CNN [5] was suggested, which includes a Region Proposal Network (RPN) that is trained end-to-end to extract feature vector and to predict both bounding box values and class scores for objects. The selective search technique, which was computationally costly and a bottleneck in the prior object detection pipeline, was replaced with this change. After being trained individually with the same loss function as in FAST-RCNN [15], the RPN finally shares convolutional features with the Fast R-CNN detector as an additional optimization. We can see the working pipeline of R-CNN based detectors in **Figure 2.2**.

On the other hand, YOLOv3 [10] and YOLOv4 [11] are single-stage detectors. YOLO does not extract region proposals but uses a Fully Convolutional Neural Network to process the full input image just once, based on the overall photographic context, which predicts bounding boxes and their respective class probabilities. **Figure 2.3** describes the workflow of one stage and two-stage detectors. The most current and effective variation of a family of one-stage detectors that process the full image in a single pass is YOLOv4. As in **Figure 2.1** YOLOv4 performs best in the term of inference speed when compared to YOLOv3 and Faster R-CNN [5], so we select YOLOv4 model architecture to train our dataset.

Here we will first describe YOLOv4 architecture, and then list our modifications made in improving YOLOv4 against our problem, i.e., small object detection.

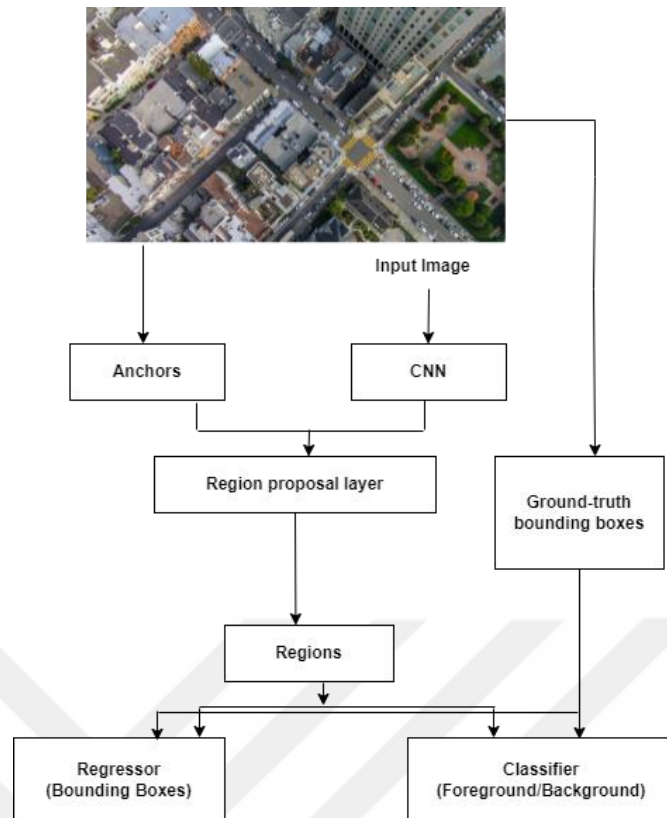


Figure 2.2: Architectural flow diagram of RCNN (two stage network) [46].

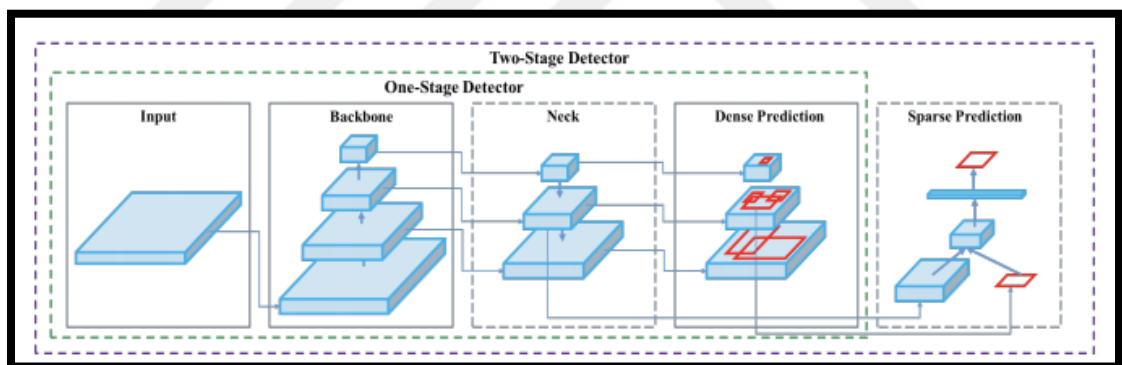


Figure 2.3: Object Detector Anatomy with several organ parts [11].

2.1.1. YOLOv4

YOLOv4 [11] outperforms other object detectors in the terms of speed and accuracy. Here we will discuss some research contributions about YOLOv4 [11]. Detection pipeline of a CNN contains three main blocks called head, neck, and backbone.

Object detector starts by taking the image as an input and then compresses the image through the network backbone to generate a feature representation for the whole image. After passing the features out of the backbone, final predictions are made off. As

multiple bounding boxes with different scales and sizes are predicted with classification, so feature layers are needed to be mixed up. The process of backbone generated feature combination happens in the neck. Finally, precise object detection happens in the head part of the network.

Object detectors, as previously said, are classified into two types: one-stage detectors and two-stage detectors. Two-stage detectors separate the tasks of object classification and localization for each object. One-stage detectors, on the other hand, make predictions for object localization and classification at the same time in a single pass. YOLO is a one-stage detector that is faster as compared to two-stage detectors.

Previous version of YOLO, i.e., YOLOv3 enhanced accuracy over prior models by adding an objectness score to the bounding box prediction, increasing connections to the backbone network layers, and making predictions at three distinct granularity levels to improve performance on tiny objects.

The following YOLOv4 object detector backbones are considered in the official YOLOv4 paper: CSPResNext50 [47] [48], CSPDarknet53 [47][49] and EfficientNet-B3 [50]. DenseNet [53] has been improved in CSPResNext50 and CSPDarknet53 to divide the feature map of the base layer by replicating it and transmitting one copy via the dense block as well as direct to the next point. CSPResNext50 and CSPDarknet53 are designed to eliminate computational constraints in DenseNet and boost learning on an unmodified version of the function map where DenseNet was designed to connect layers in CNNs to minimize the loss of gradient problem (as it is difficult to back propagate loss signals across a very deep network), improve feature propagation, enable the network to reuse features, and decrease the number of network parameters. Finally, YOLOv4 network implements CSP-Darknet53 [47][49] for the backbone network, based on intuition and a lot of experimental results.

2.1.1.1. Feature aggregation

To prepare for the detection step, the next task in the object detection procedure is to mix and match the features generated in the network. Here we will discuss about the neck part of YOLOv4. Typically, the neck components flow up and down between layers, with only a few layers attached at the convolutional network's end. YOLOv4 has investigated a few potential alternatives, the most notable of which is the Feature

Pyramid Network (FPN). [52] and PanNet [51] (Path Aggregation Network, designed for pan-sharpening)

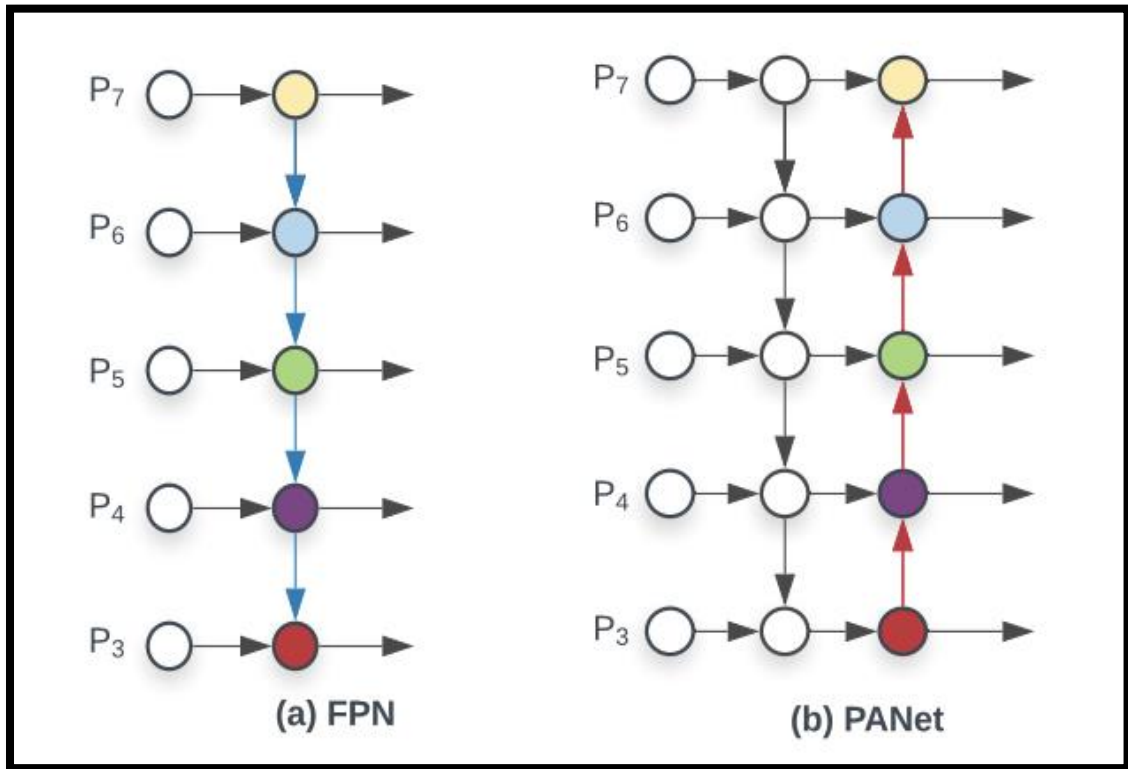


Figure 2.4: Network flow architecture – (a) FPN [52] employs a top-down approach to fusing multi-scale features ranging from level 3 to level 7. (P₃-P₇); (b) the PANet [51] adds an extra bottom-up direction to the top of the FPN [52].

2.1.1.2. Spatial pyramid pooling (SPP)

In addition, YOLOv4 [11] attaches an SPP [54] block after CSPDarknet53 to expand the receptive field and isolate the most significant features from the backbone. SPP [54] has a significantly different approach for identifying items at different sizes. In the end of the last convolutional layer, it substitutes the last pooling layer with SPP layer. The function maps are spatially broken into $m \times m$ bins of m , say, equal to 1, 2, and 4, respectively. After that, a maximum pool is added to each bin for each channel. This is a fixed-length form that can be investigated further using FC-layers.

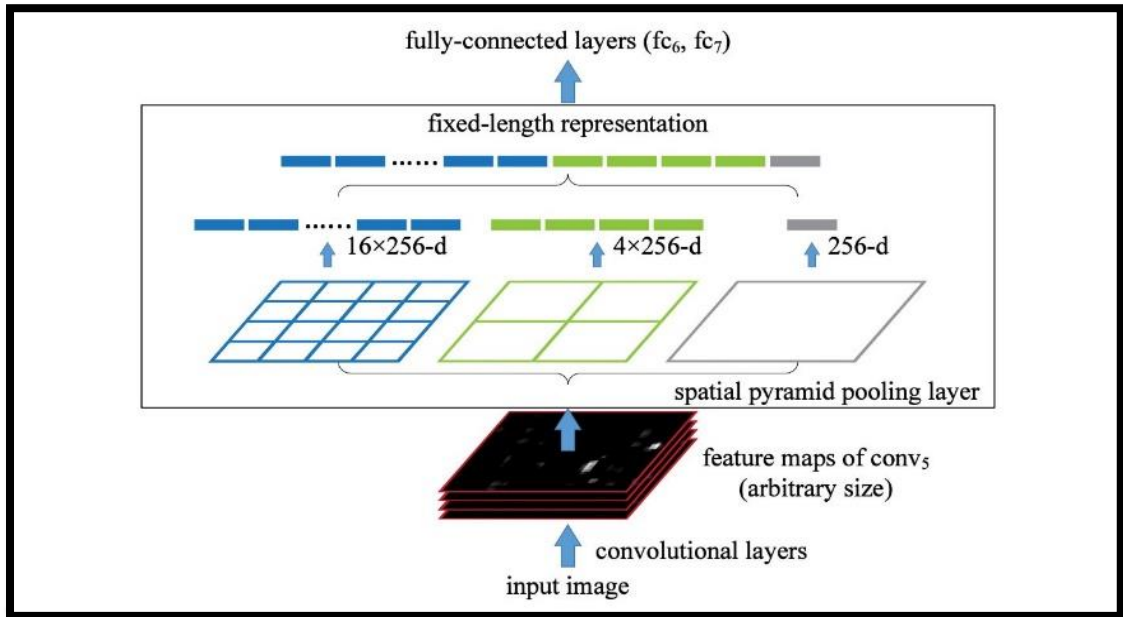


Figure 2.5: Typical spatial pyramid pooling layer in the network. In the figure conv₅ layer's filter number is 256, and conv₅ is the final convolutional layer.[54].

In YOLO neck, the SPP block [54] is adjusted to maintain the spatial dimension of the output. Maxpool is applied to the sliding kernel of dimension 1×1 , 5×5 , 9×9 , 13×13 kernels to retain spatial dimension and features maps of varying kernel sizes, which are joined together as outputs.

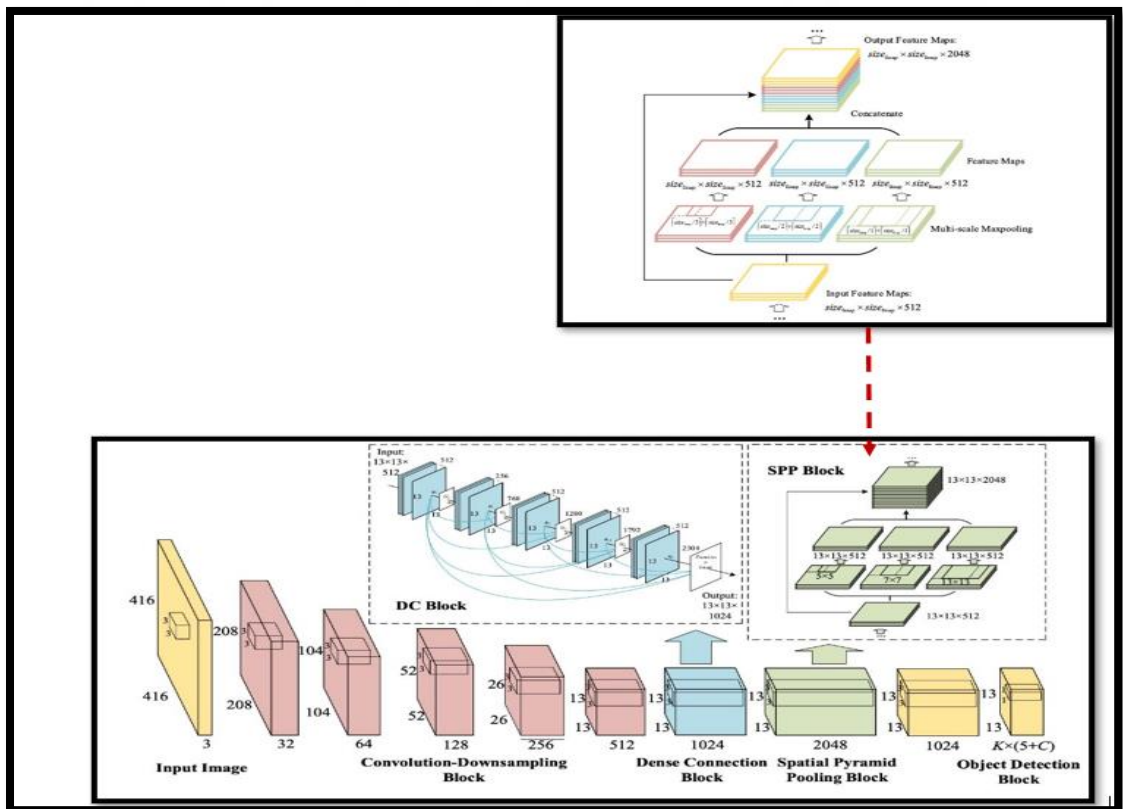


Figure 2.6: SPP block (Separate) and SSP block embedded in YOLO. [11].

2.1.1.3. Freebies

YOLOv4 [11] employs the so-called "Bag of Freebies" because it improves network efficiency without increasing output inference time. A large portion of the Freebies Bag is due to an increase in data. In YOLOv4, there are many sorts of Data Augmentation, and employing data increase in computer vision is quite significant, and it's highly suggested to get correct values out of these models. The authors employ data augmentation to increase the size of their training set and expose the model to semantic contexts that they would not have encountered otherwise.

The most recent contribution is to increase data via mosaicking, which combines four images and teaches the model to locate smaller objects while paying little attention to surrounding scenes that are not located around the object. Self-adversarial training is another unique contribution that authors contribute to data growth. SAT attempts to identify the region of the image on which the network relies the most during training, then modifies the image to hide this dependency, forcing the network to generalize to other features that can aid in detection.

Complete Intersection-Over-Union (C-IOU) loss is also used to update loss function by YOLOv4 [11] developers. In comparison to the normal IOU loss, the C-IOU loss introduces two new ideas. The first idea is to calculate the distance between the actual and predicted bounding box center points, which is called central point distance. The second is a comparison between aspect ratios of the genuine bounding box and the predicted bounding box, hence we can measure the quality of the predicted bounding box using these three metrics.

2.1.1.4. Specials

YOLOv4 [11] employs so-called "Bag of Specials" techniques, because they bring dramatic improvements to inference time but a large boost in efficiency, making them worthwhile. The Activation function, which transforms features as they pass through the network, was one of the specialties. With traditional activation mechanisms like Relu [55], it can be hard to get the network to push feature development to their full potential. Research has therefore been done to build functions that slightly enhance this method. A new activation function is proposed, Mish [56], which can be identified as:

$$f(x) = x \tanh(\text{softplus}(x)) \quad (2.1)$$

Where soft plus is another activation function and is given by $\ln(1 + e^x)$. Many studies involving performance comparison concludes MISH among best activation and the reason behind is the smooth soft plus function that returns MISH as better performer among others.

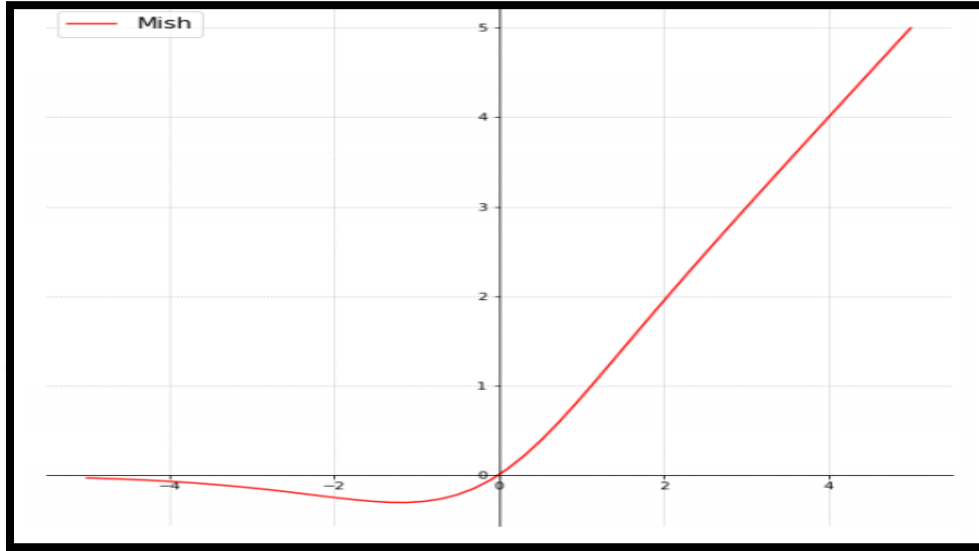


Figure 2.7: Mish activation function graph.

Experiments have shown that Mish appears to perform better than both Relu and Swish [57], along with other typical activation functions in many deep networks across demanding datasets. The non-monotonic function property of Mish [56] also helps to retain small negative values, thus stabilizing the gradient flow of the network. The most widely used activation functions, such as RELU [55]:

$$f(x) = \begin{cases} 0, & \text{if } x < 0 \\ x, & \text{otherwise} \end{cases} \quad (2.2)$$

and Leaky Relu:

$$f(x) = \begin{cases} 0.01 x, & \text{if } x < 0 \\ x, & \text{otherwise} \end{cases} \quad (2.3)$$

fail to retain negative values since they provide little or no output for negative inputs, and thus most neurons die out. Besides its good performance, one drawback of Mish activation is its high computational cost as compared to the RELU activation function.

The authors additionally employ the Distance Intersection-Over-Union (D-IOU) in Non-Max-Suppression (NMS) to identify the estimated bounding boxes. The network can predict numerous bounding boxes around single object, and it would be ideal to

efficiently find the optimal one for batch normalization. The authors additionally utilize Cross Mini-Batch Normalization (CMBN) because conventional Batch Normalization cannot operate if the batch-size is set to tiny, with the premise that it can be operated on GPUs with lower relative processing capacity. Many batch standardization techniques require multiple combined GPUs to be executed.

Finally, YOLOv4 also uses the Drop-Block regularization [58] in which different portions of the picture are hidden from the first layer. It has proven to be a powerful regularization method for object recognition. Drop-Block is a strategy that forces the network to learn features that it cannot learn otherwise. It removes contiguous regions from a layer's feature map rather than independent random units. For example, in **Figure 2.8** below, a contiguous block of $(n \times n)$ size is dropped from the feature map. The algorithm takes two inputs as parameters the $n \times n$ *block-size* and the γ , which means number of activations to be dropped. The motivation behind this is that the network should be able to distinguish either the front or the back of the car and correctly label it as a car.

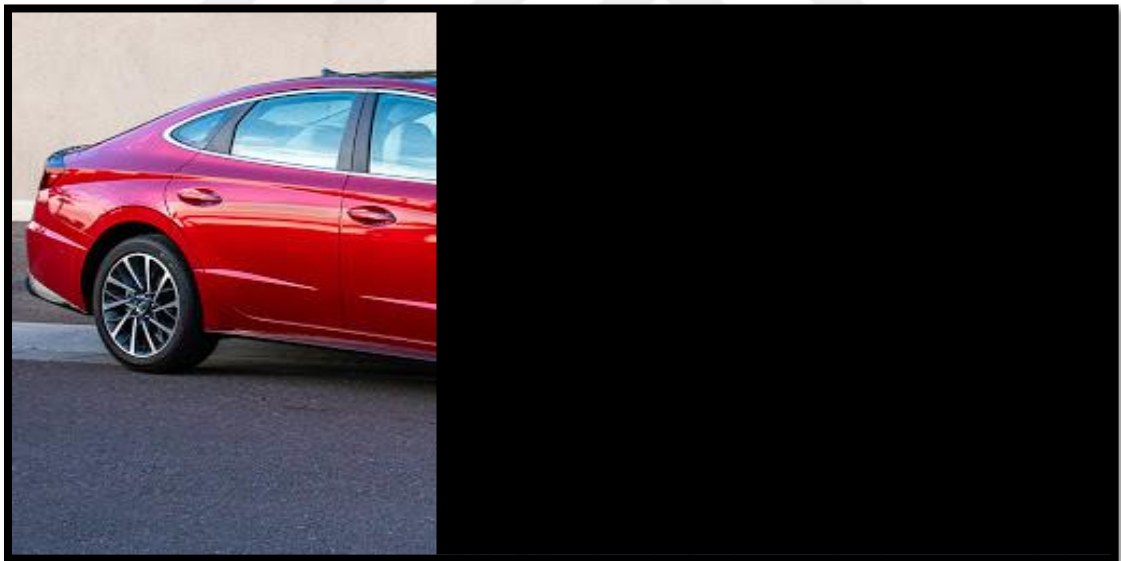


Figure 2.8: An example case (Drop block regularization), half object is appeared [58].

2.1.2. Improved yolov4 for Aerial Object Detection

YOLOv4 [11] outperforms its ancestor object detectors (YOLO 9000, v1, v2, v3) and other detection algorithms like RCNN and Faster RCNN. However, Yolo authors and related research articles have chosen and evaluated their work performance on COCO benchmark [59] which is a rich dataset in both qualitative and quantitative aspects. More than 200,000 images and 80 object categories are included in the COCO trainset,

validation, and test sets. However, if we consider object detection in aerial images then there is lack of such dataset in comparison with COCO. Hence small object detection is a challenging task for which state-of-the-art object detectors are yet to prove their applicability and high level of accuracy.

Analyzing the available datasets, we discover Visdrone to be a cutting-edge dataset in the field of UAV object detection and tracking. This dataset comprises four separate tasks: object detection, single object tracking (SOT), multi object tracking (MOT), and crowd computing. However, we solely consider the object detection task. In dataset the object dimension to image size ratio is quite low, due to which many object detectors fail while featuring out small objects. Furthermore, because drones lack high-end GPU infrastructure, so visual object detection is extremely slow. As a result, those object detection algorithms are required which can provide real-time and accurate results on existing embedded devices for drones. We intend to sort out this problem by proposing changes to the YOLOv4 [11] architecture that increase mean average precision (mAP) while maintaining real-time inference speed. Our goal is to set up object detection models that are accurate and real-time for embedded devices mounted on unmanned aerial vehicles UAVs.

The CSPDarknet53 backbone, the Mish activation function, and the FPN-PAN network are all used by the YOLOv4 network. CSPDarkNet53 [47][49] is a hybrid of YOLOv3 [10], Darknet53, and Cross Stage Partial Network (CSPNet). The CSP [47] connection is very basic and may be used with any neural network. The concept is that half of the output signal follows the main path (which provides more semantic information due to the huge receiving field) and the other half follows the bypass path (preserves more spatial information with a small perceiving field). As a result, these CSP blocks in the Darknet-53 backbone increase CNN learning while consuming less memory and performing fewer calculations. Each of these CSP modules shrink the image size by its half. Therefore, if input image dimension is 608×608 then at the final layer of backbone, feature map size will be $416/32 = 13 \times 13$.

2.1.2.1. Architectural modification

While sorting out the problem of accurate small object detection, we did modifications in the neck part of the model, where we pass the output of the fourth CBM block into an up-sampling layer while setting the up-sampling factor to 4 and pass this obtained up-

2.2. Deep Neural Network Optimization for Real-Time Execution

This section particularly focuses on how to make our detection models operate in real-time by applying certain techniques which includes TensorRT Optimizations, Multi-Threading and Parameters reduction applied on detection network.

2.2.1. Targeted embedded platform

Beside our work on software side, our final goal is a successful deployment of Object detectors and trackers on targeted embedded platform. As this research work is mainly focused on Wide Area Surveillance (WAS) and related data is UAVs Imagery so our deploying devices are UAVs which mainly have limited capacity available in terms of weight and power for processing captured data. Thanks to Nvidia for their excellent research in compact hardware platforms for embedded devices especially Jetson boards. We choose Jetson AGX Xavier, a smart embedded platform that can operate on 30 Watts of power. It is comprised of 8 CPU cores, 512 Cuda and 64 tensor cores which can be managed to operate in real-time if algorithms can be designed in efficient way so that they can efficiently consume available resources. On the other hand, Deep Learning models require much memory and processing cores to operate. To let them work in efficient manner, deep networks can be compressed using model slimming and pruning techniques. Mainly such techniques are used when results are required in real-time. Here we will discuss some of those approaches and their work-pipeline to attain real-time promising results

2.2.2. Multi-threading

Multi-threading also helps in increasing speed on detection especially when live feed from cameras is coming as input to deep models. Mainly it helps on CPU cores where the processing pipeline is not distributed as in GPUs. Hence temporarily virtual threads are created on multi core CPUs to improve processing time. A typical detection pipeline consists of three main steps: preprocess camera feed, Inferencing and post-processing after getting inference output (classes and bounding box). Explaining more precisely, first camera frames are pre-processed like reading frame from camera, transforming frame into cv.MAT format, resizing them according to model input and then feed them into the network so all preprocessing is done in one thread. Similarly, after model return detection results, again some post processing is required like NMS, plotting bounding

boxes on each object and re-scaling of box coordinates etc. So, all these postprocessing is done in third thread. Finally, after implementing multi-threading, we manage to increase FPS.

There are two implementations of detection pipeline at running time when calculating FPS. One implementation is single pipeline in which first, video frames are initialized from file or camera, second, frames are converted in to input layer dimensions in our case 416 x 416. third, after forward passing through whole network, detections and corresponding bounding boxes are embossed in image if display is needed; otherwise, we write the class and bounding-box information in the detection file.

The second implementation is to divide these steps in three different CPU based threads. Thread (video capture) will just grab frames from file and store. 2nd thread (inference) is reading image and make image ready to be load on network, this thread take image from thread process it and send to the network while network return results in three sub parts (i.e. image tensor, classes, confidence score and bounding boxes) to the queue thread (Drawing) which do post processing like scaling up boxes for localization and then draw or write in the files. Workflow of both implementations are clearly explained in **Figure 2.10** and **Figure 2.11**.

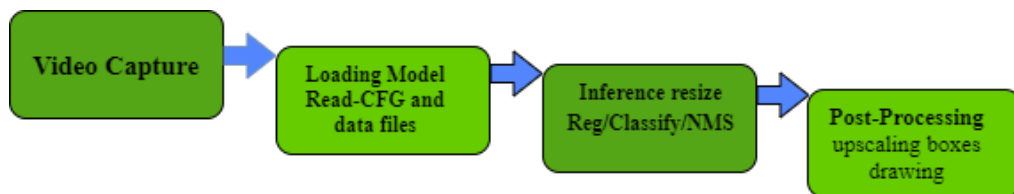


Figure 2.10: Single Threaded pipeline workflow.

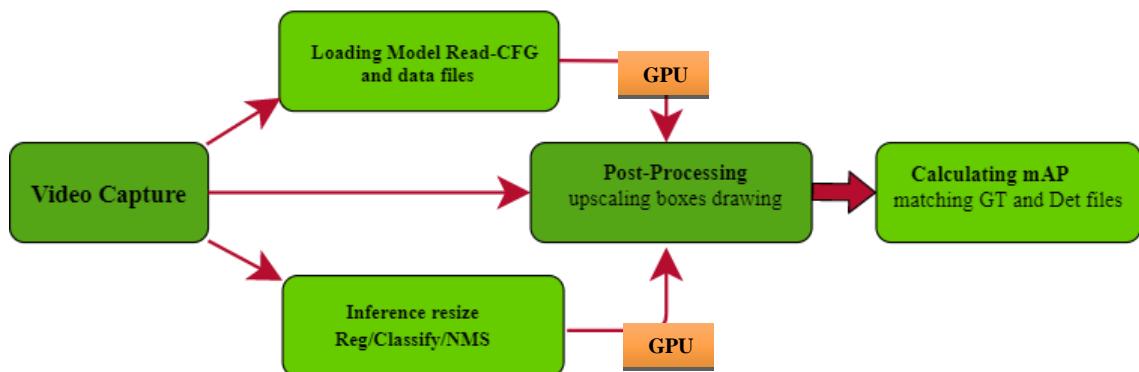


Figure 2.11: Multi - Threaded pipeline workflow (Aero head shows the flow of data between 3 independent working threads).

We also compare the performance of threaded implementation under different system configurations i.e. RTX 2080TI server and embedded computer Jetson AGX Xavier. Results in **Table 2.1** and **Table 2.2** reveal that multi-threading implementation yields 2x faster speed performance than Single threaded pipeline. It also reveals that multi-threaded approach works better with a higher number of CPU cores. For instance, in **Table 2.1** FPS didn't increase on Xavier machine having 8 cores as compared to the **Table 2.2** for RTX 2080TI Server having 32 cores.

Table 2.1: FPS Comparison among Single-Threaded VS Multithreaded Pipeline on RTX 2080TI.

RTX 2080Ti	YOLOv4	Modified YOLOv4	YOLOv3
Single Thread FPS	43	43	40
Multi Thread FPS	80	80	85

Table 2.2: FPS Comparison among Single-Threaded VS Multithreaded Pipeline on AGX Xavier. Here FPS didn't increase due to a smaller number of CPUs cores in Jetson AGX Xavier (8 CPU Cores) as compared to previous table for RTX 2080TI Server (32 CPU Cores).

AGX Xavier	YOLOv4	Modified YOLOv4	YOLOv3
Single Thread FPS	15	15	17
Multi Thread FPS	16	16	22

2.2.3. Tensor RT (TRT)

Deep neural Networks are based on certain layers or operations like convolutional layer, nonlinear activation functions and pooling layers etc., that are mainly used for deep feature extraction, providing nonlinear feature representation, and filtering out more expressive features respectively and a set of finally fully connected layers are deployed.

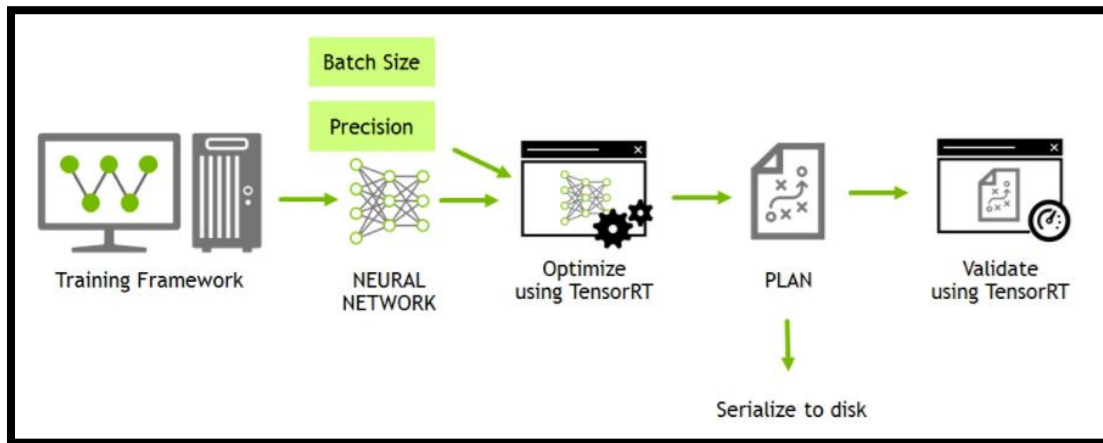


Figure 2.12: Tensor RT workflow pipeline [1].

Deep models are trained using one of the common deep learning frameworks like TensorFlow, PyTorch and Keras etc. These frameworks implementation is based on parallel processing on GPUs using Nvidia Cudnn library that provides many atomic operations like convolution, matrix multiplications and so on, written in Cuda language and provide enhanced speed while training. But still, this model optimization is not enough when we deploy our models on embedded GPUs and devices due to their limited resources. So here another library called Tensor RT helps a lot to port model to its optimized form. TensorRT has nothing to do with training stage of the model but is used just to optimize inference performance of the input model.

It contains two parts, one is Optimizer which optimizes model to less memory occupied graph storage, which is done for once and later it is used. Here we must fix some parameters like batch size (examples passed at once during a single inference) and Precision like FP32, FP16, INT16 and INT8, which is fixed at this level to optimize the model parameters. This finally generates the optimized model called TRT plan which runs on runtime Tensor RT engine for production on embedded device or clouds etc.

Importing model to Tensor RT is easy as it supports onnx format which support all type of deep learning frameworks like TensorFlow [27], PyTorch [62], Theano [63], Caffe [65] or MXnet [66] etc.

2.2.3.1. TensorRT optimizations

With TensorRT, we can do mainly 3 types of model conversion for optimized format i.e. FP32, FP16 and INT8 quantization. We will explain and implement these methods on our base detection model finetuned on Visdrone dataset. Briefly in TensorRT, once

we import model after defining its structure, we must go for these several steps towards optimization.

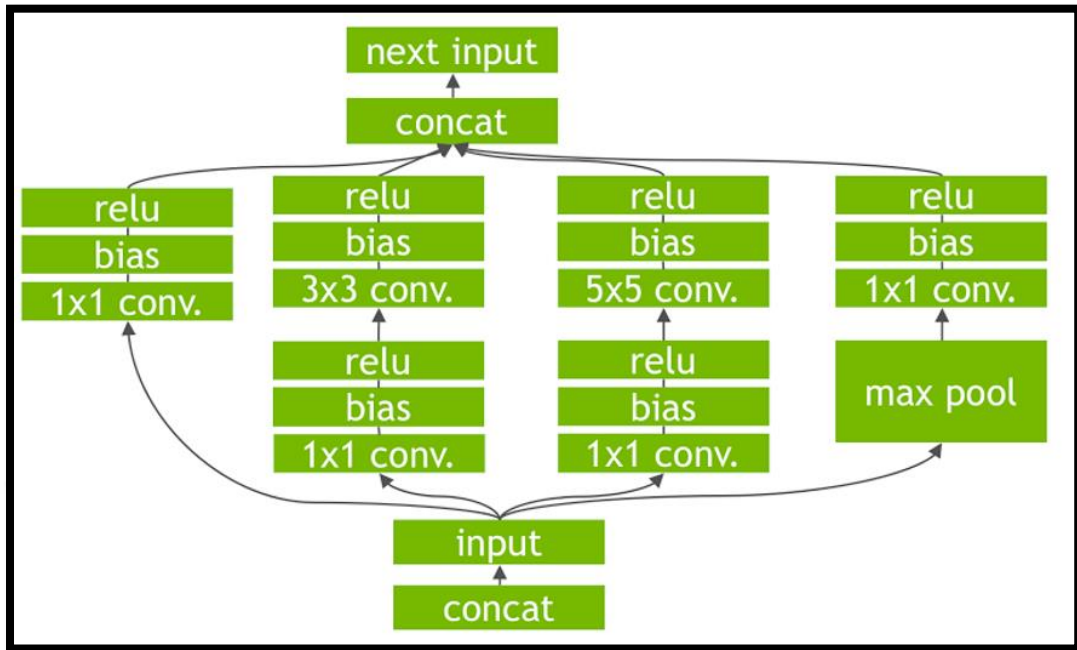


Figure 2.13: Vertical fusion Input – Un Optimized Graph [1].

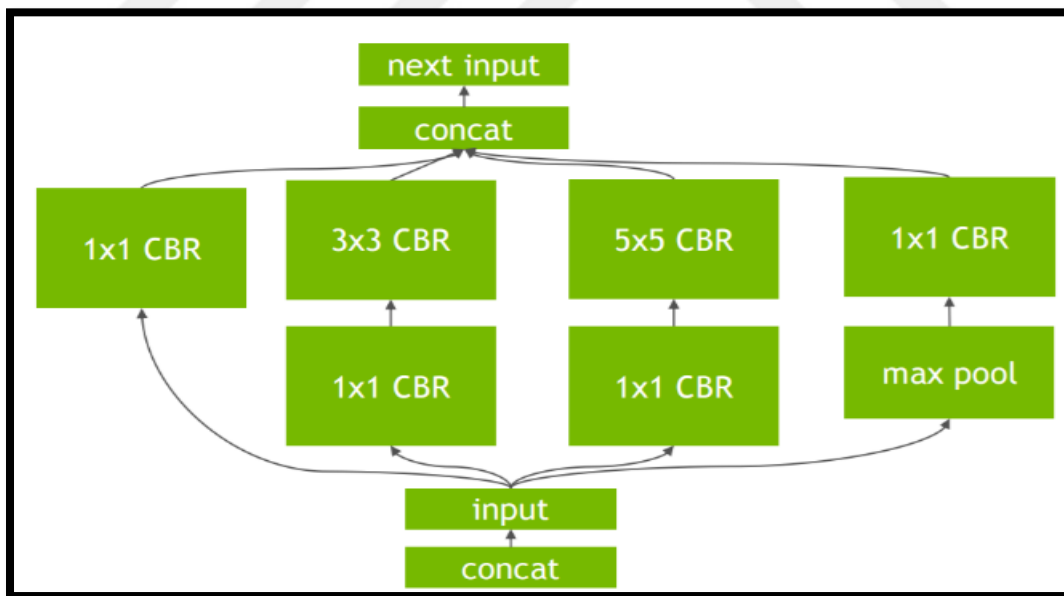


Figure 2.14: Vertical Fusion graph – Semi Optimized Graph [1].

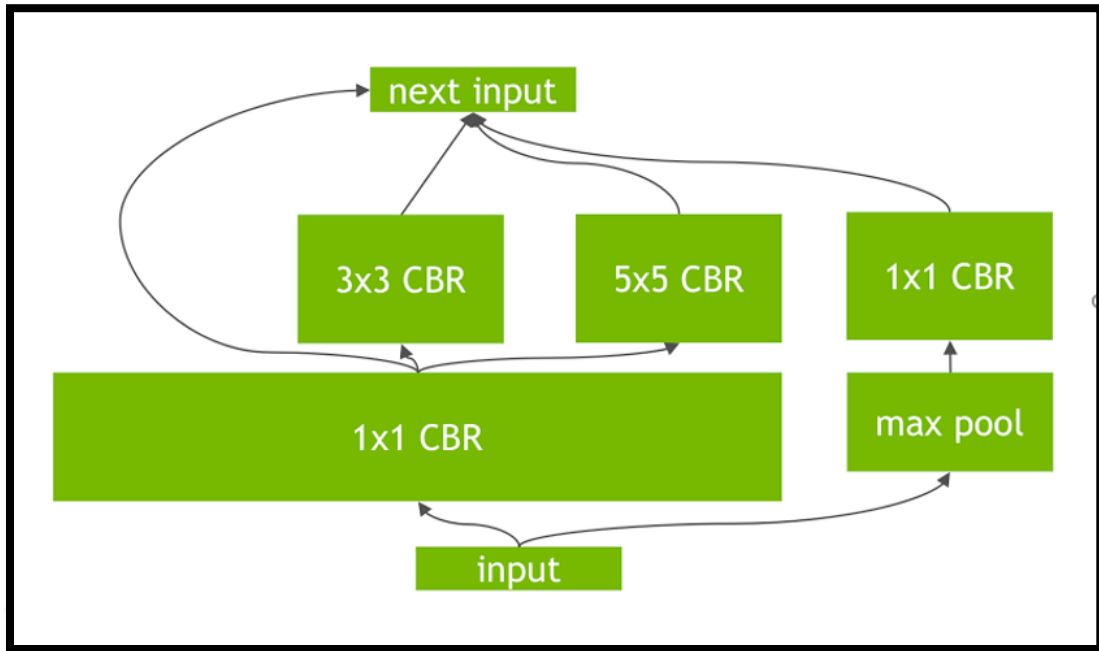


Figure 2.15: Vertical + Horizontal Fusion Optimized graph [1].

2.2.3.2. Layer and tensor fusion

Referring to above figure of un-optimize and optimized graph, on left hand side (**Figure 2.13**) we have three different layers Conv, Batch, Relu but in tensor RT version we combine and merge it as one layer (**Figure 2.14** which will be faster than three separate layers when called. This is called vertical fusion. Another case we have 3 1x1 convolution layers which have the same input, this could also be merged in one wide convolution known as horizontal fusion (**Figure 2.14**). Another concept in TensorRT is called elimination of concatenation layers. In the **Figure 2.14** the 3 layers output is going into one layer. We can see that instead of performing the same operation multiple times (**Figure 2.15**) and using concatenation operation each time, we define the layer output once in a fixed memory location, but it will be updated multiple times by other layers if needed. Hence TensorRT fuses such layers which are unnecessary for better memory manipulation and find an optimal path in inference graph.

2.2.3.3. Precision calibration and reduction

Usually while training at very high precision, Floating Point FP32 accuracy is used which provides very high exponent power in the range from i.e. -3.4×10^{38} to $+3.4 \times 10^{38}$. This level of accuracy is not necessary for all parameters of the network. So, we

may have to test different precision levels to attain speed performance. We can also downgrade operations to integer level like INT8 or INT16 optimization mode but with recalibrating our model weights which again be done with Tensor RT by passing some recalibration flags on the dataset. Tensor RT will run inference several times and try to find the weights range and then re-calibrate them to let entire network model to work properly. Once we get successful, FP16 tensor cores on GPU work very fast in computation which leads to faster FPS and less inference time.

2.2.3.4. Automatic selection of best kernels

Another approach towards optimizing the network is choosing best operations. For example, there are many ways to implement convolutions like via matrix multiplications, via Fourier transform or vinaigrette algorithm etc. and they have different implementations, tensor RT library will run each layer on a particular GPU and will select best implementation. That is why batch size and input size is fixed. As we fix our input size, so we can run convolution with different optimized algorithms. Hence tensor RT chooses best implemented form of deep learning model with respect to target platform or GPU.

2.2.3.5. Static or dynamic mode

Referring to TensorFlow ML framework, there are two TensorFlow based TRT modes: static which is a default mode and dynamic mode. If the flag (specified in the TensorFlow API as dynamic op) is set to False, static mode is allowed; else, dynamic mode is triggered. The primary distinction in between these two methods is that in static mode, TensorRT engines are built offline, whereas dynamic engines are generated during execution (by TrtGraphConverter.convert). If we have a graph with unknown feature shapes, such as camera frames with different resolutions, we can use dynamic mode. Even though TensorRT requires all shapes to be fully described, the dynamic mode enables us to have variable shapes inside our model. TF-TRT creates a new TensorRT engine for each input submitted to the model in this mode. For instance, suppose we can build an image classification network that operates on image data of any size and shape $[m \ n, 3]$. If we first transmit a few of images to a model with the dimensions $[8, 224, 224, 3]$, a fresh TensorRT engine with these dimensions will be constructed. The initial batch will take longer to complete than usual during this

moment. If we submit more images of the similar shape [8, 224, 224, 3] in the future, the already generated engine is used automatically with no additional latency. Submitting a batch of a various shapes, on the other hand, will entail the creation of a new engine for just that shape. The `max_cached_engines` parameter will be used to track the amount of engines that are saved for every `TRTEngineOp` inside this graph at any given moment. Static mode, on the other hand, does not aid in post-training measurement (INT8 calibration). If users attempt to use static mode for INT8 calibration, conversion switches to dynamic mode.

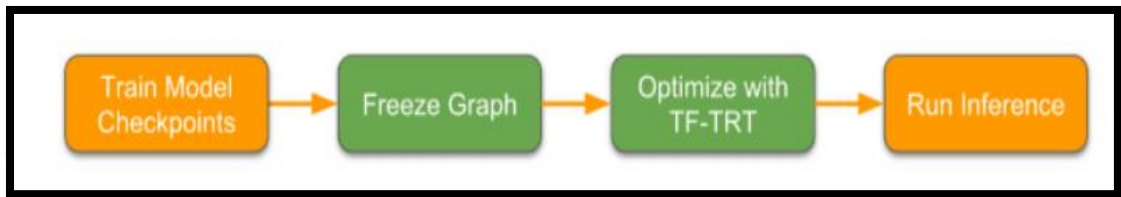


Figure 2.16: Block diagram of model conversion pipeline to TRT format.

2.2.3.6. Variable Batch Sizes

TensorRT uses the input batch size as one of the metrics to select the most capable CUDA kernels. When `dynamic_op` is set to `true` in both (TF 1.x and TF 2.0), the batch size is specified as the first dimension of the inputs and is calculated by the input shapes throughout execution.

2.2.3.7. Regulation of Minimum Node Numbers in TensorRT Subgraphs

TensorRT optimizes TensorFlow subgraphs provided by a subset of operators. If the subgraph contains a small number of operators, then it might not be effective to start a TensorRT engine for that subgraph as opposed to running the original subgraph. By using the `minimum_segment_size` argument, we can control the size of subgraphs that are to be optimized. When this option is set to `x` (3 by default), TensorRT engines are not generated for subgraphs with nodes fewer than `x`. It is critical to increase the minimum segment size to avoid creating very compact TensorRT engines (i.e., employing a very small number of layers). This will help protect the tiny TensorRT engines from future overheads and can get around any possible faults that occur from such engines. The original document report [1] shows that the default value of 3 gives most models the best result.

2.2.3.8. Memory management

TensorRT keeps track of weights and GPU activations. Each engine in the *TRTEngineOp* cache has a scale that is generally equal to the size of the weights. TensorRT uses TensorFlow allocators to allocate memory, hence all TensorFlow memory settings frequently refer to TensorRT. As instance, if the TensorFlow Session config parameter *config.gpu.options.per.process.gpu.memory* fraction is assigned as 0.3, TensorFlow will receive 30% of the GPU memory for all internal functions, including with TF-TRT module and TensorRT engine. This means that if TensorRT requests more memory than TensorFlow has available, TensorFlow will crash. TensorRT may employ algorithms that require no more than the maximum size of workspace in bytes; however, the overall workspace requirement of all TensorRT procedures may be less than the maximum workspace size, i.e. (TensorRT might not having such algorithm that requires more workspace). TensorRT just distributes the required workspace in such situations rather than determining what the user needs.

2.2.3.9. Quantization-aware training

TF-TRT is also used to transform INT8 inference models that have been trained with quantization. The error from quantized weights and tensors to INT8 is represented during train period, enabling the model to update and minimize the loss.

The approach for quantization-aware training is as follows: Before training the model, the TensorFlow graph must be updated with quantization nodes as usual. Quantization error will be described by the quantization nodes by clipping, resizing, normalizing, and un-scaling tensor values, enabling the model to adjust to the error. We have the option of using fixed quantization ranges or making them trainable variables. The question here is how a 32-bit floating-point precision model, which represents billions of different numbers, can be reduced to an 8-bit integer with only 256 possible values. In deep neural networks, weights and activations often have a limited range of values. We can maintain good accuracy if we concentrate our valuable 8 bits just in that range, with only a little rounding error. TensorRT quantizes using 'symmetric linear quantization,' a scaling mechanism from the FP32 to the INT8 range (which in our case is -127 to 127 for symmetry preservation). If we can determine the range of values for each network intermediate tensor, we can utilize that range to calculate the tensor with high precision.

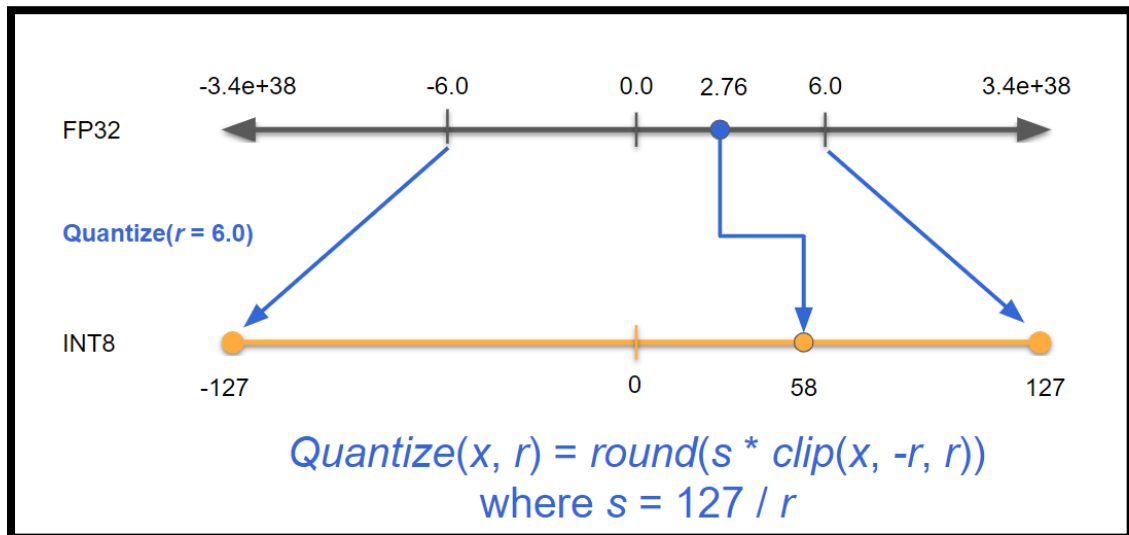


Figure 2.17: x is the input, r is the tensor floating point range, and s is the scaling factor in INT8 [1]. The above equation accepts x as an input and returns an INT8 quantized value.

The INT8 inference is modelled as precisely as possible during training. This indicates that a TensorFlow quantization node should not be inserted in sites where quantification will not be performed during inference (due to a fusion that has already occurred). TensorRT commonly merges operation patterns like Conv > Bias > Relu or Conv > Bias > Batch-Norm > Relu, therefore inserting a quantization node between each of these ops, as stated under the fusion category, would be inappropriate. It is advised to quantize nodes after activating ops such as Relu. If a required quantization range is lacking, TF-TRT can generate an error, allowing us to add that range to our graph and repeat the procedure. Alternatively, quantization nodes can be automatically added in the correct positions in our model using a method like *tf.contrib.quantize*, however this is not assured to model inference accurately using TensorRT, which can have a detrimental impact on our performance.

As our target platform is embedded GPU boards so our TensorRT transformed models will be tested on Nvidia Jetson AGX Xavier board. In **Chapter 3, Section 3.2**. FPS VS Accuracy tables will be plotted for better understanding of best techniques.

2.2.4. Deep model compression by parameters reduction

Another very significant approach to make light weight models in constructing real-time system is Parameter Reduction or Pruning. Presently, it becomes a particular field to study behaviors of deep learning models and then tweak their structure either by

reducing their parameters or transforming them to operate in mixed precision mode. All such techniques are performed before the deployment of ML models, especially on such machines where computation resources are limited such as embedded platform or fast pace working pipeline is needed. ML Engineers thoroughly investigate about trainable parameters and after performing several experiments to reduce trainable parameters, a customized model is generated, which can speed-up the inference time with zero or very low accuracy loss. We comprehensively investigate our object detection model (YOLOv3 and YOLOv4) by its structural and parametric aspects as described in **CHAPTER 2, Section 2.1.1**. Later we perform several pruning techniques which are explained in up-coming section under this chapter.

2.2.4.1. Training of base model

The goal of basic training is to create a huge model with great precision. Its accuracy and parameter selection are important for further sparse training, and it can also be used for knowledge distillation, which is another technique to learn pruned model from base model. It is not suggested to neglect basic training to directly prune or sparse the model. In our case we choose Visdrone 2019 object detection dataset [2] for training. Visdrone is a challenging dataset due to small objects and limited data examples; so, for basic training, we select YOLOv3 [10] and YOLOv4 [11] models as a base model to train on Visdrone dataset to get more accurate and precise large model in terms of parameters and memory size.

2.2.4.2. Sparse training

A sparse matrix is a matrix containing many zero entries. While applying operations on matrices, the resultant matrices sometimes contain many zero entries or a number very close to zero like 0.00001. Applying operations like multiplication on each of such entries and storing them in a memory lacks performance in real-time during inference. Therefore, if we remove such entries and transform matrices to compact form, we get gain in memory size and speed in training and inference. On the other hand, removing all these zeros will probably have an impact on network accuracy or performance of the base model. Experiments reveal that applying sparsity during training causes a drop in accuracy; but after passing sparse network through several epochs, accuracy is regained to the same level as it was before. While implementing sparsity training, a sparsity

factor is set which can be considered as a threshold for model parameters. One important observation to get benefit from sparse training is that; the higher the sparsity factors the lower should be the learning rate. Sparsity factor should be set in this way because it leads to better results, but of course it claims more time to regain the accuracy after instant drop. This inverse relation between learning rate and sparsity factor can be written as follows:

$$\alpha \approx \frac{1}{A} + x, A > 0 \quad (2.4)$$

Where α is a hyperparameter representing learning rate and A is the sparse factor and x is the increment factor to sparsity and is always positive.

Figure 2.18 shows difference between pre and post specified tensors which results in faster execution and inference time.

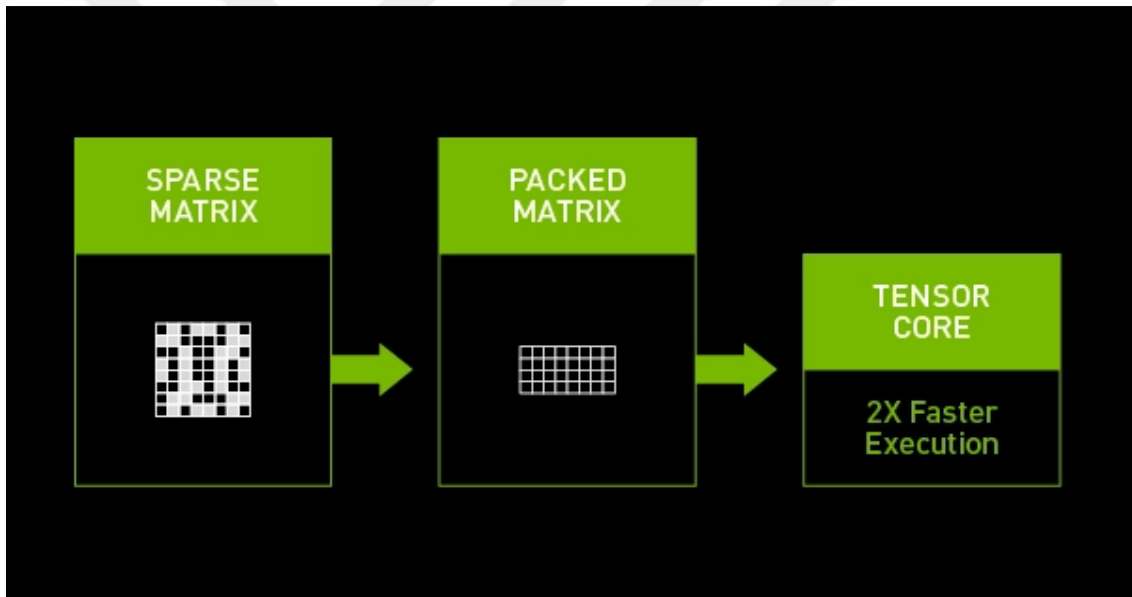


Figure 2.18: The effect after sparsifying the given tensor, resulting 2 times faster execution [64].

From our experiments in training model with sparsity factor, we observed base model accuracy drops significantly (**Figure 2.19**) but later after long trade-off between mAP VS epochs, it restores mAP of the initial base model, hence one can obtain extra sparse matrices and reduce memory operations with sparsity training [67].

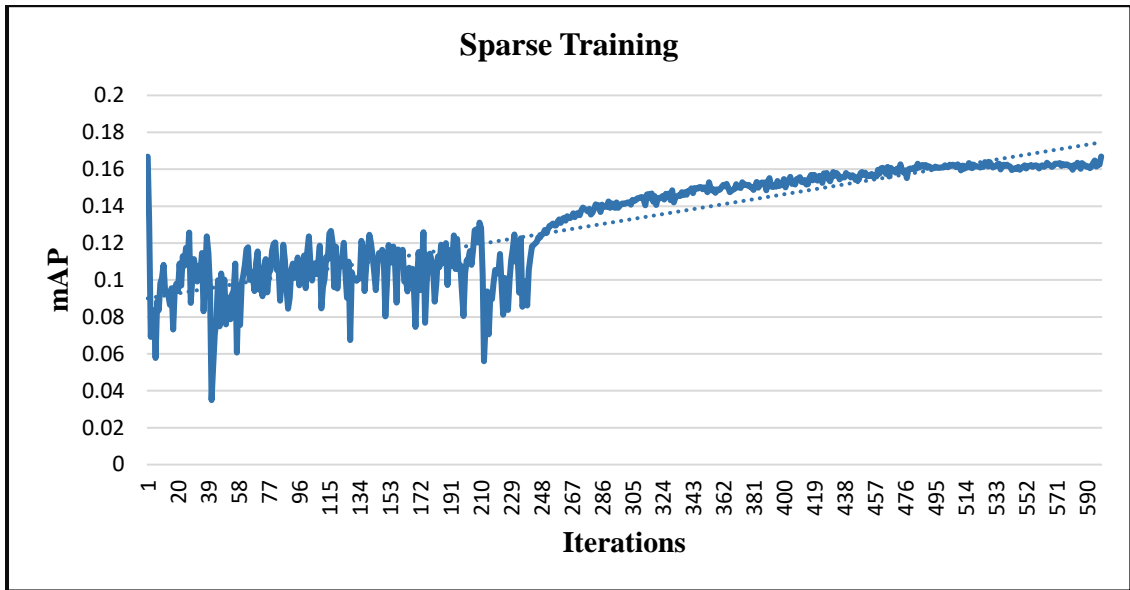


Figure 2.19: Accuracy restoration while sparse training.

2.2.4.3. Channel pruning

After restoring accuracy by applying sparsity on base model, channel pruning is applied, which reduces model parameters by cutting some connections and replacing or reconnecting them. Channel pruning is a kind of limited or protected pruning approach, meaning it has less pruning ratio compared to other methods. So base model architecture is modified by not cutting the layers; instead, it simply prunes the channels. Normally, there are five groups of 23 shortcut connections in YOLOv3, based on the add operations. To ensure that the two input dimensions of the shortcuts are the same after applying channel pruning, this pruning avoids the difficulty of dimensional processing by not cutting the layers directly connected to the shortcuts. Also, it produces a higher pruning rate on the channels, which aids in the reduction of model parameters. Channel pruning is especially powerful if it is used in addition with other pruning strategies. In our experiments channel pruning is used in slim pruning and results in reduction of 14 million parameters of base YOLOv3 model (**Table 2.8, Table 2.9**). After applying, even though number of parameters and model size are decreased, but at the same time drop in accuracy is also observed. So, we try this process with different percentage numbers. Finally, we obtained reasonable accuracy by applying fine-tuning on the generated model after converting it to darknet weights file with corresponding cfg file.

2.2.4.4. Layer pruning

This approach is an extension of the prior channel pruning strategy. It analyzes the preceding CBL of each shortcut layer, sorts the gamma averages within each layer, and prunes the smallest layer. When a shortcut structure is cut here, a shortcut layer and the two convolutional layers next to it are also cut. To preserve the YOLOv3 structure's integrity, all layers linked to the shortcut layer must have the same channel number. To match the feature channels of each layer that are connected by a shortcut layer, we loop through the pruning masks of all connected layers and perform an OR operator on all these pruning masks to create a final pruning mask for these layers.

Note that only shortcut module in the backbones is considered here. However, the YOLOv3 architecture includes 23 shortcuts. Removing 8 shortcuts cuts down 24 layers, whereas cutting 16 shortcuts means dropping 48 layers.

In our experiments, we consider 6 shortcuts to be pruned. For one shortcut, 2 CBL (Convolution - Batch Normalization - Leaky Relu) block cut-off from the network, which means ($6 \times 2 = 12$) layers are erased. This reduces around 18 million parameters. Finally, fine tuning of this model makes it more precise and accurate in the terms of mAP. We performed certain experiments for Visdrone data. Experiments results are mentioned in results section in tabular form showing parameters reduction, improvement in inference time and corresponding accuracy loss. However, after fine-tuning the network over augmented data we achieve 4% higher mAP on Visdrone test-dev dataset as compared to the base model, which is explained in detail in **Section 3.4**.

2.2.4.5. Network slimming

Most of the modern CNNs use batch normalization [68] as a common strategy to achieve quick convergence and superior generalization performance. After a convolutional layer, it is common practice to add a BN layer with channel-wise scaling/shifting parameters. As a result, the parameters inside the BN layers can be used directly as the scaling factors required for network slimming. It has the significant benefit of causing no network overhead. In fact, this could be the most effective method for determining important scaling factors for channel pruning.

Training with sparsity-induced regularization at the channel level results in a model with multiple scaling factors close to zero. Thus, channels with scaling factors with

values close to zero are pruned by eliminating all input and output connections, as well as the weights associated with them. All layers prune the channels using a global threshold defined as a given percentage of all scaling factor values. For example, by setting the percentile threshold to 0.7 will prunes 70% of channels with lower scaling factors. However, when the pruning ratio is large, pruning may cause some temporary accuracy loss. However, the subsequent fine-tuning procedure on the pruned network can help compensate loss in accuracy. **Figure 2.20** shows the flow-chart of network slimming [69].

In YOLOv3 there are plenty of BN and Conv layers called CBL blocks, hence the whole pruning technique is applied to these layers to get maximum advantage.

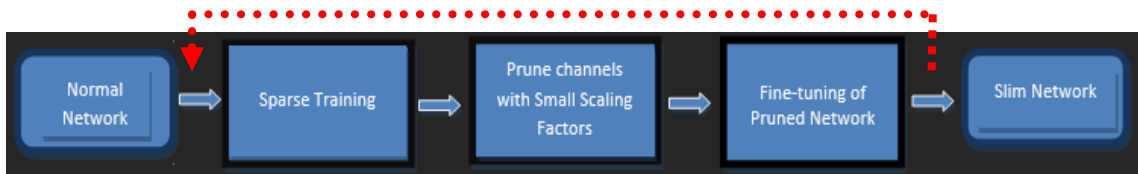


Figure 2.20: Network Slimming step by step workflow.

As shown in **Figure 2.14** above, the idea behind slimming is to introduce a scaling factor for each channel, and then multiply it with the output of the channel. Then jointly train the network weights and these scaling factors, and finally directly remove the channels with small scaling factors and fine-tune the pruned network. In particular, the objective function is defined as:

$$L = \sum_{(x,y)} l(f(x, W), y) + \lambda \sum_{\gamma \in \Gamma} g(\gamma) \quad (2.5)$$

Among here, (x, y) represents the training data and labels, W are the trainable parameters of the network, and the first term is the training loss function of the CNN. $g(\gamma) = |\gamma|$ denotes L1 regularization on scaling factor, and λ is the balancing factor of the two terms. In the experiments, L1 regularization is chosen. The sub-gradient descent method is used as an optimization method for the unsmooth (non-derivable) L1 penalty term. We can also replace the L1 penalty term with a smooth L1 regular term and avoid using sub-gradients at unsmooth points.

Technically this approach is based on two main points:

- 1- To use the scaling factor γ as the pruning factor in in batch normalization, that is, the lower the γ , the less essential the associated channel is, and it can be pruned.

- 2- To reduce the size of γ , add an L1 regularization term about γ to the target equation to make it sparse, this allows automatic pruning during training, which was not seen in earlier model compression strategies.

Utilizing these above points there are further approaches having slightly differences to prune the network but reveals better results.

2.2.4.6. Normal pruning

In normal pruning following are the steps implemented on YOLOv3 architecture:

1. Find the BN layer's associated index that has to be cropped.
2. The gradient obtained by the L1 regularization is added to the gradient of the BN layer before each backpropagation.

The trainable scale factors inside BN layers are used as channel significance indicators. Channel-wise sparsity training is used to efficiently distinguish significant channels from unimportant channels by imposing L1 regularization on γ . In YOLOv3, BN layers follow convolution layers and normalize convolutional features using small batches approach and is given by formula:

$$y = \gamma \times \frac{x - \bar{x}}{\sqrt{\sigma^2 + \epsilon}} + \beta \quad (2.6)$$

Where σ and \bar{x} denotes mean and variance in mini batch and γ and β represents scaling factor and bias respectively. Training objective with sparsity training is composed of both object detector (YOLO) and L1 regularization given by $L = loss_{yolo} + \alpha \sum f(\gamma)$.

3. Determine and set the cropping rate by retrieving the absolute value of the BN layer's parameter of the cropped layer into a list and sorting from small to large. As an instance, if the cropping rate is 0.8, the cropping threshold equals the value of the 0.8 quantile in the list.
4. Set the γ of the channel below cropping threshold to 0, then check for the clipped mAP value. The bias value β should be non-zero.
5. Make a new model structure and combine β into running mean computation in BN layer, next to the convolutional layer.
6. Finally, build a new model file.

2.2.4.7. Optimized normal pruning

This technique is very similar with normal pruning, explained earlier. The only difference is that here shortcut layer in the architecture is pruned while avoiding cutting-off the entire layer.

Shortcut Layer is basically a skip connection, determining that output of the previous layer and any other layer is merged. As an instance, in yolo architecture definition file, if the *from* parameter is set to -3, then it indicates that the shortcut layer's output is obtained by adding feature maps from the third preceding layer backwards from the shortcut layer. The first three steps in this pruning technique are the same as in the normal pruning. After setting the clipping threshold, we will

1. Extract the channels whose parameters are less than the clipping threshold; and if the channel γ of the entire layer is lower than the threshold, then to avoid the entire layer from being clipped, keep the channels with larger γ 's value in the layer, so we should set threshold according to the layer parameters.
2. The mask of the shortcut layer is then merged, and the union strategy is adopted easily because we didn't cut the whole layer from original architecture, instead we choose such layer where moderate channel values are present, which avoids whole layers to be clipped.
3. Verify the mAP of the model.
4. Finally, we will compare the number of trainable parameters and inference speed.
5. If it is accurate enough in comparison to base model, generate new cfg and save the weights, otherwise fine-tuning can be applied.

2.2.4.8. Shortcut pruning

Shortcut pruning is pretty like normal pruning in terms of technique. It analyzes the CBL of each shortcut layer, sorts the Gamma values of each layer, and selects the least values to prune convolution layers with shortcuts. In implementation, mask of first convolutional layer is used in each shortcut group. Hence in total 5 masks are used for convolutional layer pruning for 5 shortcut layers present in architecture. This does not reduce number of layers but decrease number of channels to 1, instead of cutting them entirely.

2.2.4.9. Layer channel pruning

This strategy is the joint combination of Channel as well as Layer pruning, meaning cutting layers and channels at the same time. Layer and channel pruning can be used to reduce the model's width and depth. It may be used in both simultaneous and iterative pruning modes to create a good recipe towards parameter reduction. This approach of simultaneous layer pruning, and channel pruning is also applied on our base models to facilitate the comparison of pruning effects.

The experiments related to all pruning techniques are provided in **Chapter 3, Section 3.3**.

2.3. Multi Object Tracker (MOT) Deployment

As our main goal is to design a real-time multi object tracker, so after applying all the techniques proposed in previous chapters, we select best of object detection models to embed in a tracker, that can perform in real-time and maintain a reasonable trade-off between mAP (Mean Average Precision) and FPS (Frames Per Second). Based on our all simulations we prefer YOLOv3 base layer channel pruned model due its good mAP and lower number of layers. Its mAP is 30 and recorded frame per seconds are 23 on jetson Xavier. Turning it to mixed precision mode like FP-16 even raises FPS above 40 which is quite enough to perform tracking in real-time. **Figure 2.21** in **Section 2.2.4.11** shows overall performance of all models being used in experiments. Anyone of these real-time models can be selected to use in multi object tracking pipeline either in Tracking by Detection or Joint Detection and Tracking (JDE) [41] based Multi Object Tracker for tracking purpose.

2.3.1. The deep sort tracker

For tracking by detection, we implement Deep Sort algorithm [70] which is based on two different CNN pipelines. The first pipeline is object detector which performs localization and classification. The second part consists of two sub algorithms (Deep Appearance Descriptor and Kalman Filter) which help to generate final tracklets. For appearance descriptor another CNN based feature extractor is used which turns each localized object in to a well discriminating feature embedding, making it well compatible with cosine appearance metric used in evaluation process. The Kalman filter helps in measuring the velocity of the detected objects on the eight-dimensional state

space $(u, v, \gamma, h, x', y', \gamma', h')$ that contains the bounding box center position (u, v) , aspect ratio γ , height h , and their respective velocities in image coordinates. This assumes constant velocity motion and linear observation model, where the bounding coordinates (u, v, γ, h) are observations acquired from previous pipeline working behind it as an object detector (YOLO).

2.3.1.1. Assignment problem

To resolve the relationship between projected Kalman states and newly received measurements, the authors construct it as an assignment problem that can be solved using the Hungarian [72] algorithm. The method incorporates motion and appearance information by combining two relevant metrics. The (squared) Mahalanobis distance [71] is used to include motion information between projected Kalman states and newly appeared measurements. Mahalanobis distance [71] produces information about possible object locations based on motion that are especially useful for short-term predictions. This distance between freshly acquired measurements and estimated Kalman states is computed as follows:

$$d^{(1)}(i, j) = (d_j - y_i)^T S_i^{-1} (d_j - y_i) \quad (2.7)$$

where (y_i, S_i) is the i -th track distribution's projection onto measurement space and d_j is the bounding box of acquired detection. The cosine distance [73], on the other hand, helps to determine appearance information, which is particularly important for recovering identities after long-term occlusions when motion is less discriminative. The second metric computes the shortest cosine distance [73] in appearance space between the i -th track and the j -th detection and is given by:

$$d^{(2)}(i, j) = \min_k \left\{ 1 - r_j^T r_k^{(i)} \mid r_k^{(i)} \in R_i \right\} \quad (2.8)$$

Here r_j is the appearance descriptor for detection d_j and $R_i = \left(r_k^{(i)} \right)_{k=1}^{L_i}$ is the collection of the last hundred ($L_i = 100$) related appearance descriptors, for each track i . Hence both metrics i.e. Mahalanobis and Cosine distance complement each other by covering different aspects of the assignment problem. To build the association problem, both metrics are combined by a weighted sum as follows:

$$c_{i,j} = \lambda d^{(1)}(i, j) + (1 - \lambda) d^{(2)}(i, j) \quad (2.9)$$

where $d^{(1)}$ represents the Mahalanobis distance between freshly acquired measurements and estimated Kalman states and the second metric, $d^{(2)}$, quantifies the shortest cosine distance in appearance space between the i -th track and the j -th detection. λ is a hyperparameter which can be used to control the influence of each metric on the combined association cost between the i -th track and j -th detection. During experiments λ was set to 0, as stated by authors that they found setting $\lambda = 0$ is a reasonable choice when there is significant camera motion. In this way, only appearance information is used in the association cost term.

2.3.2. Joint object detection and embedding (jde) tracker

JDE [41] tracker is explained in detail in literature review (Section 1.1.6), which is faster as compared to other discussed object trackers. Typically, two stage detection pipelines (tracking-by-detection) are used in which first stages produce localized targets and then for each of these targets in the second stage, during the association step, identified objects are allocated and linked to known trajectories. The main difference among existing pipeline JDE pipeline and previous approaches i.e. separate detection and embedding (SDE) is illustrated in the **Figure 2.21**.

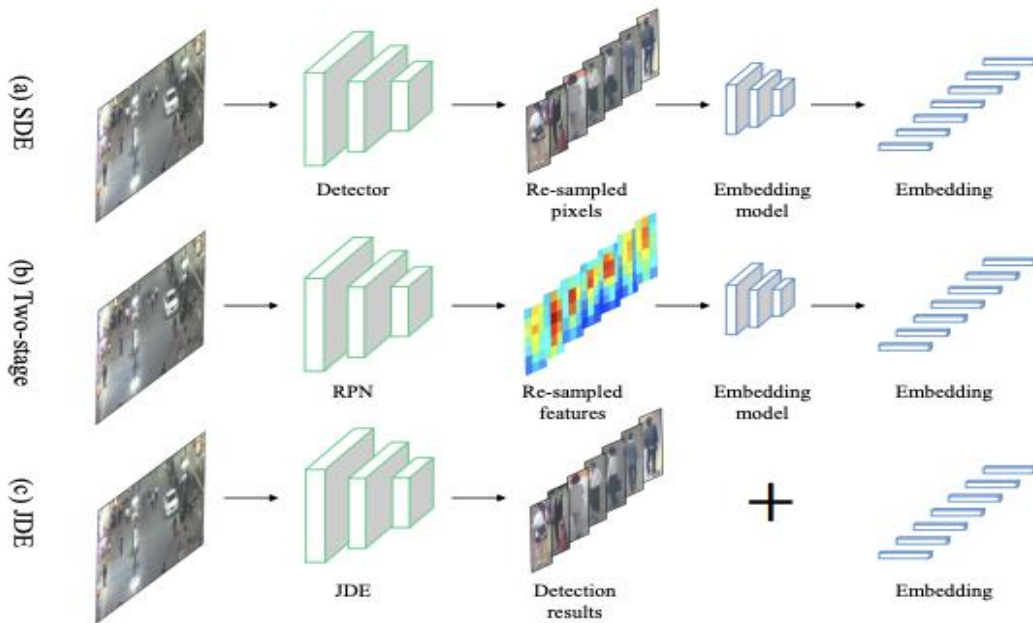


Figure 2.21: shows a comparison of (a) the Separate Detection and Embedding (SDE) model, (b) the two-stage model, and (c) the suggested Joint Detection and Embedding (JDE) model [41].

2.3.2.1. Detection pipeline

In general, the detection branch is identical to the regular RPN, however there are two differences. First, the authors modified the quantity, scale, and aspect ratio of anchors in their work to adjust to the object, i.e., pedestrians. They set all anchors to a 1:3 aspect ratio. The total number of anchor templates A is set to 12, so $A=4$ for each scale, and the anchor's scale (width) spans from around 11 to 512 pixels. Second, they emphasize the significance of selecting an acceptable value for double threshold used for foreground/background assignment. The authors also establish that an $\text{IOU} > 0.5$ associated with the ground truth roughly ensures a foreground, consistent with the generic object detection, but boxes with an $\text{IOU} < 0.4$ associated with the ground truth should be considered background rather than the 0.3 applied in general settings. The foreground/background classification loss L_α and the bounding box regression loss L_β are the two loss functions associated with the detection learning objective. The cross-entropy loss is denoted by L_α and smooth loss as L_β .

2.3.2.2. Appearance embedding

The second goal is to define the learning problem, i.e. to learn a feature representation or embedding space in which occurrences of the same identity are near together but instances of different identities are far apart. An effective technique for achieving this goal is to employ triplet loss and optimize it. The triplet loss L_{triplet} is given by:

$$L_{\text{triplet}} = \sum \max(0, f^T f_i^- - f^T f^+) \quad (2.10)$$

In relation, f represents the embedding for detection d in a frame. f^T is a mini-batch instance chosen as an anchor. f^+ denotes positive sample in relation to f^T , while f^- denotes a negative sample. There are various challenges with this simplistic definition of the triplet loss. The first is the training set's large sampling space. In the paper, this issue is handled by examining a mini-batch and mining all the negative samples and the most difficult positive sample in this mini-batch and here f^+ is the most difficult positive sample in a mini-batch.

2.3.3. Automatic loss balancing

Each prediction head's learning objective in JDE can be treated as a multi-task learning problem. The overall objective is expressed as a weighted linear sum of losses from all scales and components.

$$L_{total} = \sum_i^M \sum_{j=\alpha,\beta,\gamma} \omega_j^i L_j^i \quad (2.11)$$

where ω_j^i are loss weights, ($i = 1$ to M , $j = \alpha, \beta, \gamma$) and M denotes the number of prediction heads. α , β and γ can be determined as three different loss weights from multitask learning problem. In multi object tracking, these three losses can be listed as object detection, bounding box regression and prime triplet loss (to predict feature embeddings for tracking) respectively.

2.3.4. Online association

The JDE model provides the bounding box and apparent characteristics of each target for a particular video, allowing the correlation matrix between the newly discovered target's apparent characteristics and the current target trajectory to be determined. Detected targets can be linked to historical tracking trajectories using the Hungarian method. For trajectory smoothing and target position prediction, the model uses the Kalman filter. Cosine similarity calculates appearance affinity, and Mahalanobis distance is used to calculate motion affinity. The linear assignment issue is then solved using the Hungarian approach. The Kalman filter updates the motion state of all matched tracks, as well as the appearance state. To update the apparent features of the linked detection and trajectory, the following formula is used:

$$e_i^t = \alpha e_i^{t-1} + (1 - \alpha) f_i^t \quad (2.12)$$

Where, f_i^t is the appearance embedding of the current matched observation, and the momentum term $\alpha = 0.9$. Finally, observations that appear in two frames in a row but are not allocated to any tracklets are created as new tracklets. If a tracklets has not been updated in the last 30 frames, it will be terminated.

2.3.5. Multi – class joint object detection and embedding (MC-JDE) tracker

From the paper it has been observed that IDF1 score, (correct detections ratio over the average of ground-truths and predicted detections) value is low, and more ID switches are detected [41], which is one of JDE's drawbacks. The author initially assumed that

was due to JDE's poor apparent feature learning, however after replacing the apparent feature extraction model with a separate learning, the IDF1 value and ID Switch did not change significantly. Finally, the author discovered that this was primarily due to insufficient object detection accuracy when more than one pedestrian crossed each other. A failure is depicted in the **Figure 2.22**. The author's future research claims to focus on improving detection accuracy when targets are overlapped.



Figure 2.22: Failure case comparison and wrong detection results when detected objects have high overlapping, results in more ID switches [41].

Among other Multi Object tracking (MOT) algorithms JDE differentiate itself by applying one stage tracking pipeline in which location and appearance embeddings are produced in a single pass and later embeddings are used to track each associated object produced from detection. However, original JDE tracking pipeline is based on Faster-RCNN [5] (two staged) object detector which is much slower the one stage detectors like YOLOv3/v4 [10][11]. Also, in original implementation JDE tracker consists of single class, i.e., person class, to learn more accurate embedding and then differentiate them to track. This makes the scope of JDE quite limited in speed and robustness. We train YOLO based object detectors on 10 different classes present in Visdrone Object Detection Dataset [61] and then use this multiclass object detection model in JDE pipeline, which improves the FPS as well as turns JDE to multiclass detection and tracking pipeline thus called Multi Class MC-JDE tracker. Note that in this implementation the embeddings are generated from the three YOLO layers present in object detection model. Previously it was extracted from the second stage of Faster-RCNN detection architecture. Hence MC-JDE can detect and track multi objects in less time and complexity.

Simulation results and comparisons for the trackers are provided in **Chapter 4, Section 4.4**.

CHAPTER 3

3. EXPERIMENTAL PART

This chapter includes all experimental details for the topics discussed under theoretical part in **Chapter 2**.

3.1. Experimental Setup for Small Object Detection

We performed several experiments on Visdrone dataset under several settings and compare results for both models. Visdrone [61] is a complex real-world dataset with objects of various sizes, viewpoints, weather, scale, and light. It includes 7215 photos for training, 548 for validation, and 1610 for testing. The images were acquired by multiple drone-mounted cameras and cover a wide range of locations and crowd densities. The dataset contains ten object categories: motor, bus, awning-tricycle, tricycle, truck, van, car, bicycle, person, and pedestrian. Training of both models was carried out on a GPU machine whose specifications are given in **Table 3.1**.

Table 3.1: Hardware specifications of machine being used for experiments.

System	Configuration
Operating System	Ubuntu 18.04
CPU	Intel Core (TM) i7-10700F CPU @ 2.90GHz
GPU	2 X NVIDIA RTX 2080 8GB Graphics card (R)
RAM	32 GB
Hardware memory	1 TB

We ran multiple trials on the dataset with several image sizes and measured mAP in each one. Experiments revealed that the modified YOLOv4 achieved 2% better mAP results than the original YOLOv4 [11] at varied image resolutions on the Visdrone object detection dataset [2] while operating at same FPS.

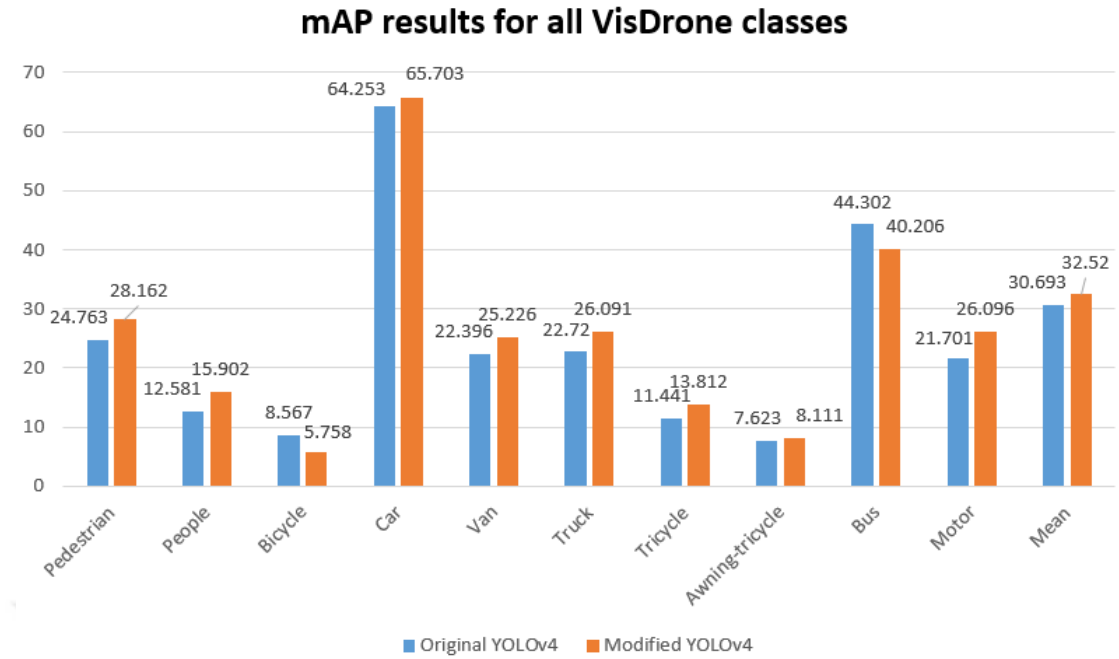


Figure 3.1: mAP results of original and modified YOLOv4 [60] for all object classes at 832x832 image resolution.

Although the training was done at 416x416 image size, the trained model's performance was observed at various image resolutions during the testing stage. The results show that the model's performance improves as the size of the test image increases, with the best performance at 832x832 image resolution. On the test-dev dataset at 832x832 image resolution, the results of the original and modified YOLOv4 [60] for each class are also shown and analyzed. The graph reveals that, except for the bicycle class, improved YOLOv4 [61] performs better almost in all other categories. As there are more cars samples in the training dataset than any other object, so both models perform best on cars.

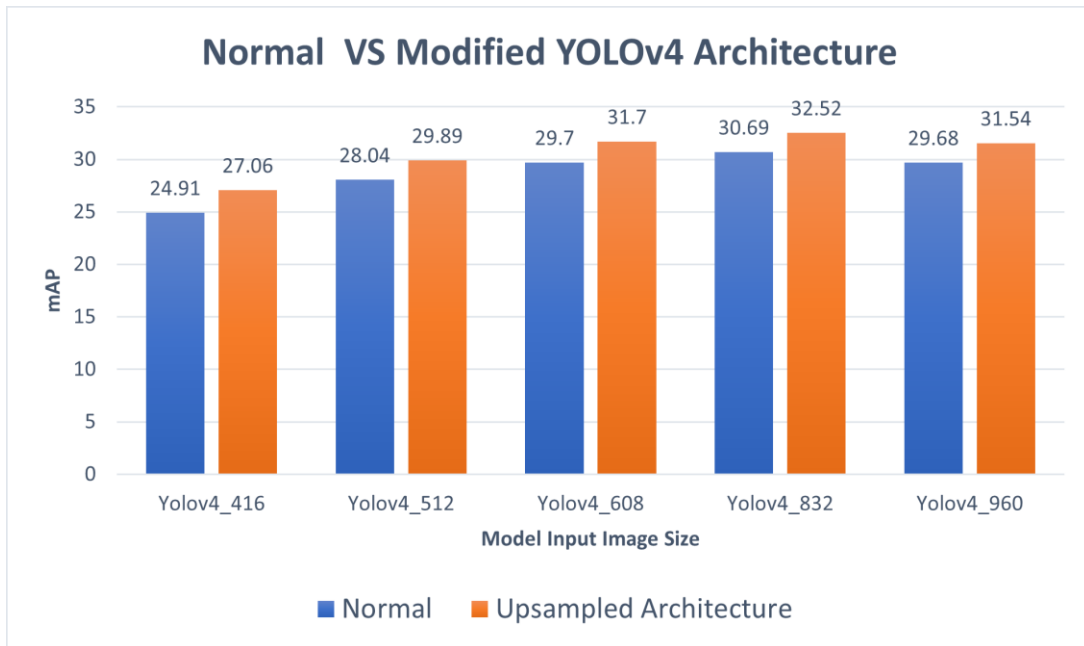


Figure 3.2: mAP calculated on original and modified YOLOv4 for different image resolutions at test stage [60].

We trained the original YOLOv4 as well as the modified YOLOv4 architecture on Visdrone data. Every input image is scaled to 416x416. All tests are carried out with an input image size of 416x416 and a learning rate of 0.0001. The reason for using the 416-image patch is to simplify and limit computations. After every 1000 epochs, the weights are saved for mAP calculation and to ensure that the model is learning properly. All trained model's results are evaluated on the Visdrone test-dev dataset at various image resolutions by using the official Visdrone DET Toolkit [61].

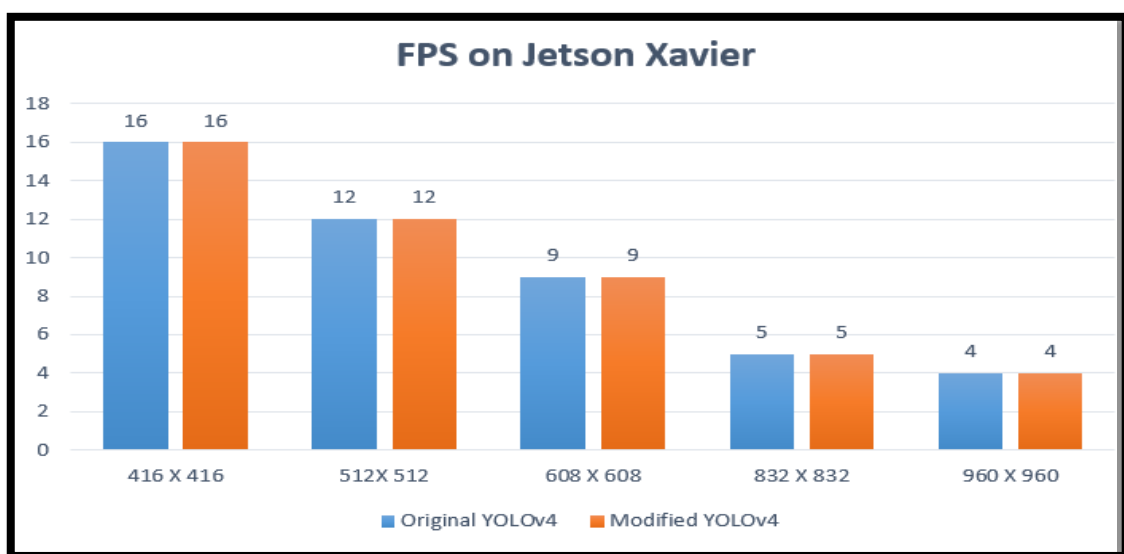


Figure 3.3: FPS results of original and modified YOLOv4 [60] for different image resolutions at test stage.

3.2. Experimental Setup for Tensor Optimizations

We take 3 model architectures for applying tensorRT optimizations i.e. YOLOv4, modified YOLOv4 [11], and YOLOv3 [10] pretrained on Visdrone 2019 object trainset. The reasons to choose these architectures are that YOLOv4 [11] is considered a state-of-the-art detector for its performance on the COCO [59] benchmark, but after tweaking the architecture an updated YOLOv4 (modified YOLOv4) also performs well on detecting small objects. YOLOv3 also contains feature pyramid network and make predictions at three different levels to recognize objects more precisely. To make them light weight and in compressed form, we consider transforming these trained models in fast mode by changing their precision state from FP32 to FP16 using the Nvidia TensorRT framework. The main reason for this is to make them more real-time while maintaining minimum impact on accuracy. **Table 3.2** presents the measured accuracy as mAP and speed as FPS of the base models whereas **Table 3.3** and **Table 3.4** shows mAP and FPS measured on their respective TensorRT mixed precision lightweight (**FP16** and **FP32**) versions. We can see that after applying TensorRT pipeline, in **Table 3.3** FPS slightly increases as network is optimized by vertical and horizontal fusion; but when we apply FP16 which reduces floating point operations to the half, FPS increased more than 2X as compared to unoptimized versions (**Table 3.4**)

Table 3.2: mAP VS FPS of original (YOLOV3, YOLOV4 and Improved YOLOv4) non optimized versions (before serializing to TRT format) evaluated on Visdrone Test-Dev Dataset on Jetson AGX Xavier.

AGX XAVIER	YOLOv4	Improved YOLOv4	YOLOv3
mAP @ 0.5	28.0	33.0	23.5
Comparing FPS	12.0	12.0	16.0

Table 3.3: mAP VS FPS on Visdrone Test-Dev Dataset with TensorRT FP32 models on Jetson AGX Xavier

AGX XAVIER FP32	YOLOv4	Modified YOLOv4	YOLOv3
mAP @ 0.5	30.0	33.0	23.3
FPS	13.0	13.0	16.0

Table 3.4: mAP VS FPS on Visdrone Test-Dev Dataset with TensorRT with FP16 on Jetson AGX Xavier.

AGX XAVIER FP16	YOLOv4	Modified YOLOv4	YOLOv3
mAP @ 0.5	29.0	33.0	23.2
FPS	30.0	30.0	37.0

3.3. Experimental Setup for Model Compression and Parameters Reduction

We choose our base model as YOLOv3, trained on Visdrone Object Detection dataset, on 10 classes. The main purpose was to produce highly accurate large and deep model before cutting any channels or layers of the model. As said earlier, sparse training is recommended prior to doing pruning. Hence, for that purpose, we did sparse training in such a way that small learning rate corresponds to higher sparsity factor and higher learning rate corresponds to smaller sparsity factor. We did sparse training for around 600 epochs. After sparse training, the new model generated is then converted into darknet weights file. This reconverted model is fine-tuned on the same dataset to regain the accuracy. **Table 3.5** provides the comparison between base model and pruned models. After pruning we can see that number of layers and trainable parameters are reduced which lead to reduce frame processing time; hence increase the speed performance (FPS) of detection pipeline. Typically reducing parameters has negative effect on accuracy. In the table, we can see that for Normal pruning mAP is badly affected; but after applying Layer Shortcut, Slim and Layer Channel pruning the mAP is increased from the base model. This is due to the fact that in such pruning techniques layers are not straightforwardly fused or cut by some percentage like in Normal pruning but only those channels whose parameters are less than the clipping threshold are removed; and if the channel γ of the entire layer is lower, then to avoid the entire layer from being clipped, we keep the channels with larger γ 's value in the layer, so we should set threshold according to the layer parameters. Moreover, Sparsity training and Fine tuning of the newly generated weights results in more precise results as compared to the base model whose mAP is 23.5 whereas layer, slim and layer channel have 31.5, 30.7 and 30.8 mAP respectively. In-detailed discussion of all pruning algorithms and steps are clearly explained in **Section 2.2.4.3** and **Section 2.2.4.4**.

Table 3.5: Table shows the results against applied pruning techniques on YOLOv3 detection network. A higher reduction in model size and number of network layers and Increase in FPS and mAP indicates a better mode.

	Base Model	Normal Pruning	Layer Shortcut (6 shortcuts)	Slim Pruning	Layer Channel Prune (6 shortcuts)
mAP	23.5	9.0	31.5	30.7	30.8
No of layers	106	106	94.0	104	88.0
Parameters	62 million	42 million	52 million	48 million	48 million
FPS	16.0	14.0	22.0	17.0	22.0

3.4. Data Augmentation

As Visdrone is a challenging dataset in terms of complex scenarios and limited data samples, very high accuracy on such dataset is not possible. Meanwhile pruning different models also lead to accuracy loss. So, we tried to augment the data in a different way particularly for Visdrone dataset or similar alike. Normally image size of Visdrone is around 1920 x 1080 and our model input size is 416 x 416. We perform two different techniques: 1) tiling and 2) zooming to increase the number of data samples for training. Zooming also helps in detecting same objects at different scales. This augmentation technique led to the significant improvement in the mAP.

The proposed method works in two stages, first it converts the original image into $n \times n$ tiles and then observe which tiles have objects; if the tiles have objects, then it saves the tile in existing trainset as well as perform zooming on the same tile. If the tile is without any object, it's neglected. Finally, after zooming the tile containing object it includes this resultant tile in existing training dataset.

3.4.1. Experimental setup

We choose YOLOv3 as a base model for training on augmented dataset. This reason was that YOLOv3 is most light, less complex, and suitable. Also, YOLOv3 pruning implementation is easily available to make it more customizable for specified task. We trained YOLOv3 Base and pruned model on augmented dataset as explained above.

Besides that, we also trained Normal YOLOv4 and Improved YOLOv4 on our augmented dataset to verify the validity of augmented dataset. All models perform much better as compared to the models trained on normal Visdrone Object Detection Dataset as shown in **Figure 3.4**.

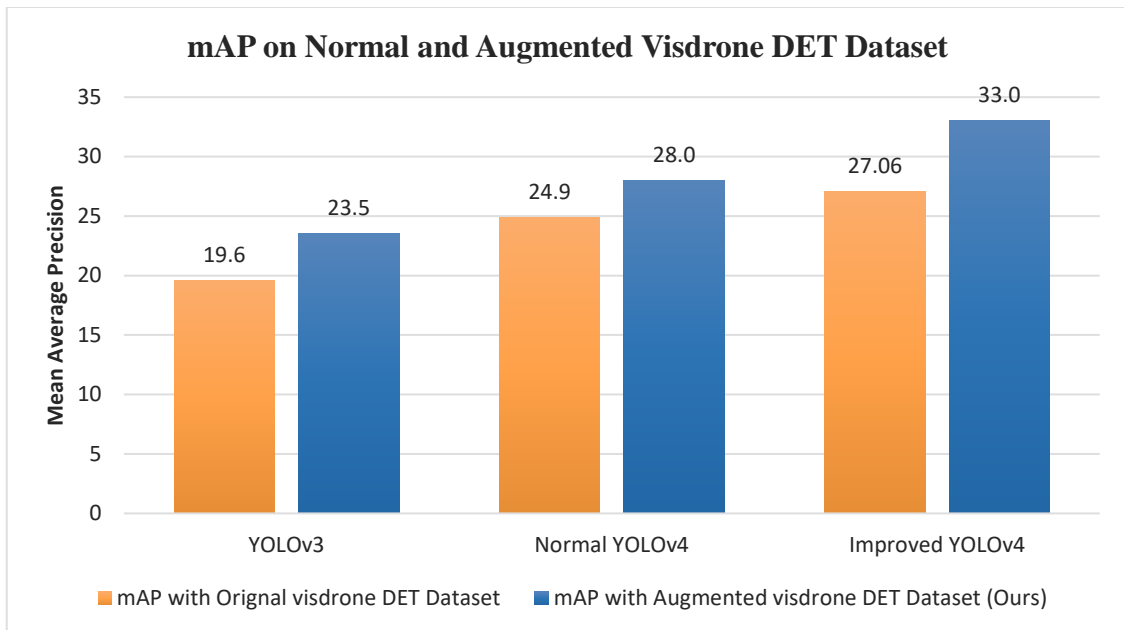


Figure 3.4: Plot shows accuracy improvement after including Augmented data (Tiles + Zooms) in Visdrone DET dataset.

CHAPTER 4

4. RESULTS AND DISCUSSIONS

4.1. Improved YOLOv4

We achieved improvement in small object detection for images captured from UAVs. In the original YOLOv4 architecture, we connected up-sampling layers, resulting in more robust and accurate features for small objects. The recommended improvements increased mAP by 2% (as observed in **Figure 4.1** when compared to the original YOLOv4 [11] model but had no effect on inference speed. The achieved result of 16 frames per second at 416x416 image sizes is still less than 30 frames per second for real-time object detection. In the next chapter, we apply TensorRT [1] optimizations and multithreading techniques to boost speed and achieve real-time object recognition.

Finally, we also find that class imbalance issue in Visdrone dataset leads to low average precision (AP) while evaluating each class separately. For example, in a case Car vs. People, Pedestrian and Motorbike, Car class has many instances as compared to other classes present in the dataset. To overcome this problem, there must be some data driven approach to overcome accuracy issue. **Section 3.4** in this report addresses the problem and its solution in detail.

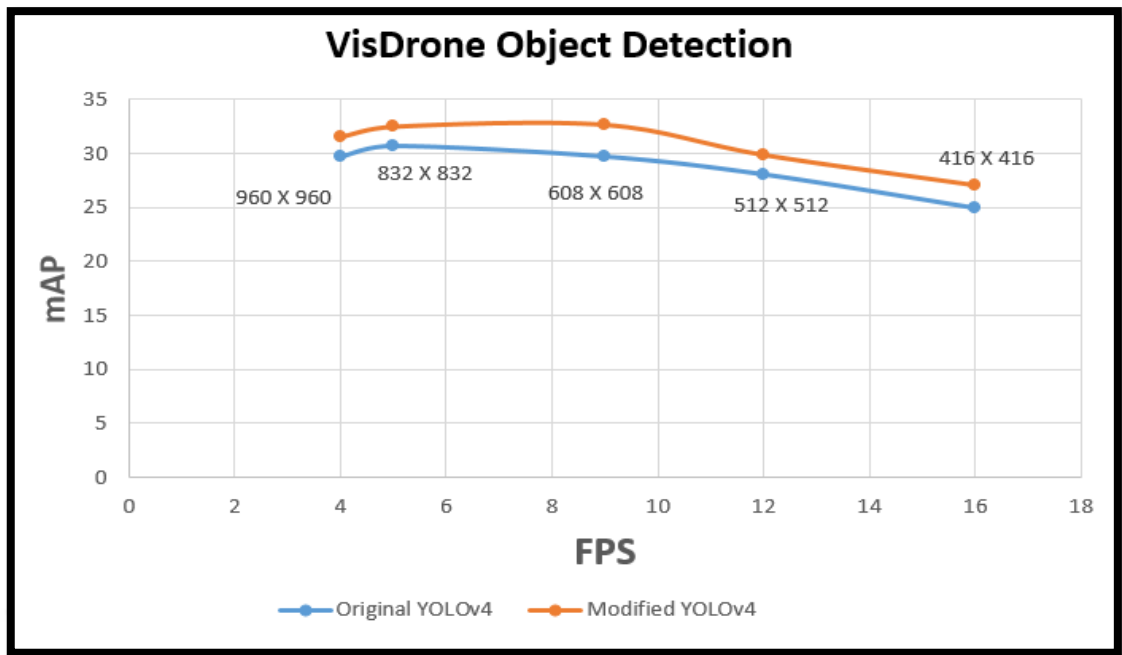


Figure 4.1: Comparison of original and modified YOLOv4 in terms of mAP and FPS for different image resolutions on Jetson Xavier [60].

4.2. Model Compression by Parameters Reduction

As our main goal is to design a real-time multi object tracker, so after applying all the techniques proposed in previous chapters, we select best of object detection models to embed in the tracker, which can perform in real-time and maintain a reasonable trade-off between mAP and FPS. Based on our all simulations we prefer YOLOv3 shortcut and layer channel pruned models with 6 shortcuts due their good mAP and fewer trainable parameters; their mAPs are 31.5 and 30.8 respectively and recorded FPS are 22 on jetson Xavier. Turning it to mixed precision mode like FP-16 even raises FPS to 55, which is quite enough to perform object detection in real-time. **Figure 4.2** shows overall performance of all models being trained and used in our experimental work.

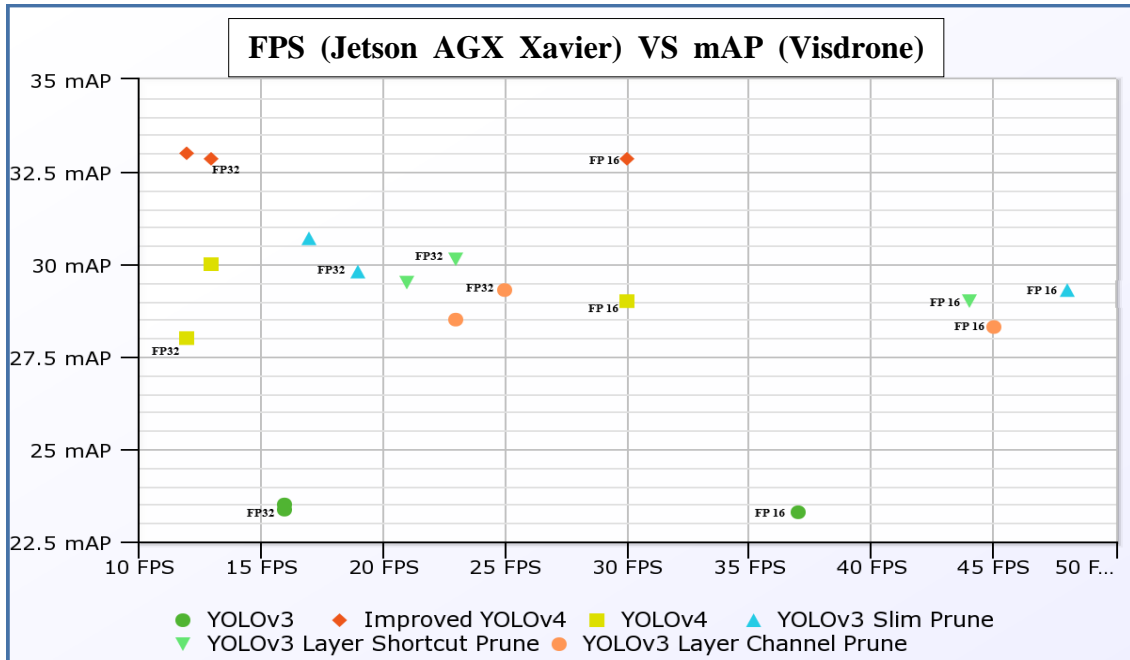


Figure 4.2: mAP VS FPS plots of object detection models; FP16 and FP32 represents corresponding TensorRT models. FP-16 models are more directed towards real-time.

After implementation and testing of numerous pruning methods, best models will be selected and embedded in Tracking by Detection or Joint Detection and Tracking (JDE) based Multi Object Tracker, for UAVs to make it more real-time for embedded platform, which in our case is Jetson AGX Xavier.

4.3. Ablation Study for Pruning Parameters

While applying pruning techniques on our base models, there are certain parameters or thresholds like pruning ratio, global channels pruning percentage and channel keep percentage per layer, based on which pruned models are further tested to get the best model while tweaking these parameters. We test pruned models while applying pruning factors with different percentages.

For normal pruning, we first pruned our model with a 0.4 threshold, i.e., 40 per cent of the model layers on average will be fused. This results in mAP loss and pruned model is to be fine-tuned afterwards to gain mAP. Finally, testing reveals model performance in terms of mAP loss and FPS gain. **Table 4.1** shows pruned model after applying Normal pruning at a 40% prune rate. As normal prune straight away cuts 40 % of all channel parameters so a high accuracy drop is observed. Similarly, for other discussed pruning techniques like a slim prune, we check performance on two different parameters i.e. **Global Channel** and **Channel keep** ratio which determines what percentage of the

total channels should be pruned and for each layer how many channels should be preserved. We roughly checked on different values of these parameters, but the most suitable results were observed with the values presented in **Table 4.2**. This results in a nearly 20% reduction in model parameters and leads to better mAP and FPS as compared to the base YOLOv3 model.

Similarly, for Layer Shortcut and Layer Channel Pruning, we prune on Shortcut layers present in the base YOLOv3 model. Note that as shortcuts are the skip connection, so removing shortcuts removes two CBL (Conv, Batch-Norm and LeakyRelu) blocks as well. This implies that the more shortcuts are pruned, the more parameters will be reduced resulting in compactness in size. Among 22 shortcuts in total, we perform experiments by considering 6, 8, 10 and 14 shortcuts with reasonable accuracy drops. Increasing the number of removed shortcuts results in a higher accuracy drop. Finally, after applying sparsity training and fine-tuning on remaining weights; the number of layers, parameters, mAP on Visdrone DET Test Dev benchmark and FPS on Jetson AGX Xavier are measured as displayed in **Table (4.3 and 4.4)**. Finally, we also applied TensorRT serialization to all these pruned models which led to being faster as compared to their base pruned version. The highest number of FPS we achieved is with mixed precision FP-16 TRT model. **Figure 4.3** shows reduced parameters and the corresponding maximum FPS of pruned models in (FP16).

Table 4.1: FPS of pruned model (Normal Pruning) on prune ratio 40. When pruned more than 40%, mAP reduces to zero.

Base YOLOv3 mAP	Prune Percentage	Trainable Parameters	Pruned mAP	Darknet FPS	TensorRT FP32 FPS	TensorRT FP16 FPS
23.5	40% prune	42 million	6.0	14.0	23.0	53.0

Table 4.2: FPS of pruned model (Slim Pruning) on two different pruning ratios.

Base YOLOv3 mAP	23.5	23.5
Slim Prune Parameters	Global Channel Prune %: 0.15 Channel keep ratio %: 0.02	Global Channel Prune %: 0.30 Channel keep ratio %: 0.01
Trainable Parameters	48 million	46 million
Pruned mAP	30.7	18.0
Darknet FPS	17.0	12.0
TensorRT FP32 FPS	19.0	20.0
TensorRT FP16 FPS	48.0	50.0

Table 4.3: FPS of pruned model (Layer Shortcut Pruning) on different prune ratios.

Base YOLOv3 mAP	23.5	23.5	23.5	23.5
Shortcut Prune Parameters	Shortcuts: 6	Shortcuts: 8	Shortcuts: 10	Shortcuts: 14
Trainable Parameters	48 million	46 million	40 million	36 million
Pruned mAP	31.5	30.7	30.13	29.0
Darknet FPS	21.0	22.0	23.0	26.0
TensorRT FP32 FPS	22.0	20.0	21.0	23.0
TensorRT FP16 FPS	35.0	37.0	44.0	53.0

Table 4.4: Layer Channel pruning experiments under two different pruning parameters.

Model	YOLOv3	6 Shortcuts YOLOv3	8 Shortcuts YOLOv3	10 Shortcuts YOLOv3
Layer Channel Prune with Parameters: Global Channel Percent %: 0.15 Layer keep ratio %: 0.01	Shortcuts: 0	Shortcuts: 6	Shortcuts: 8	Shortcuts: 10
Trainable Parameters	61 million	48 million	46 million	41 million
Pruned mAP	23.5	30.8	26.5	29.3
Darknet FPS	16.0	21.0	23.0	25.0
TensorRT FP32 FPS	16.0	22.0	22.0	23.0
TensorRT FP16 FPS	37.0	35.0	38.0	45.0

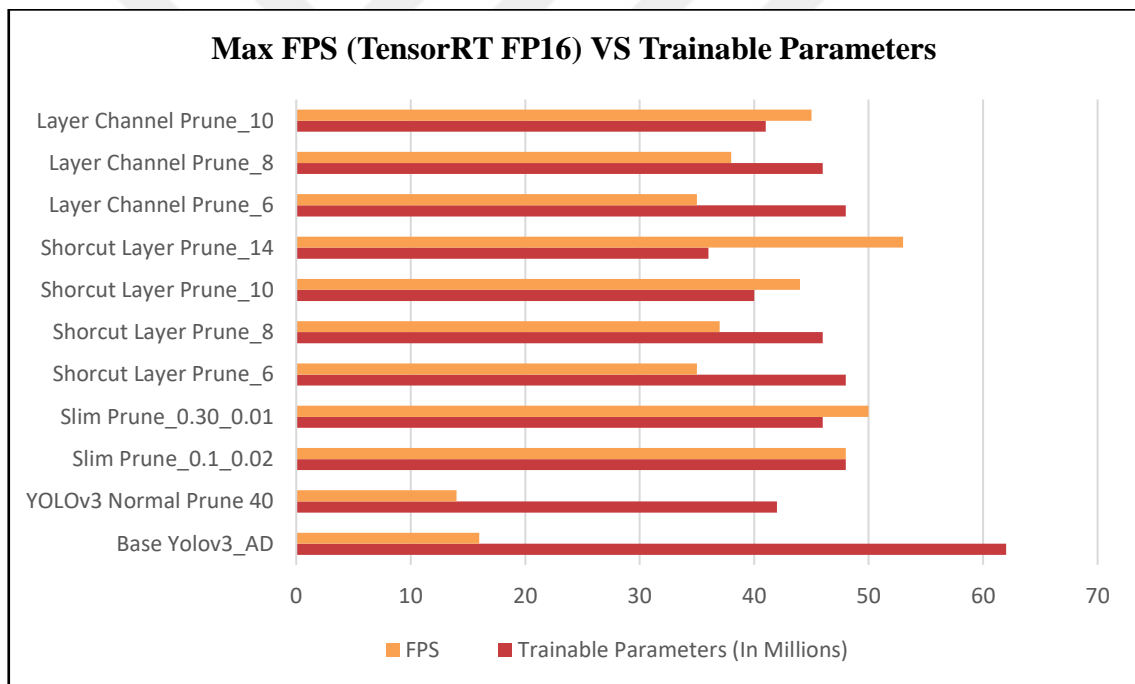


Figure 4.3: Plotting trainable parameters in millions (Red color) reduced after applying pruning techniques and corresponding highest FPS (Orange color) as compared to base models.

4.4. Multi Object Tracker (MOT) Deployment

As described in **Section 2.3.5** the detection pipeline in JDE is less accurate and contains complex architecture which results in less accurate tracking results and lower FPS respectively. We perform detailed experiments on improving object detection pipeline

in both aspects. After all we manage to produce light weight as well as more accurate detection models as compared to their base architecture by using YOLOv3 [10] which is single stage detector. After selection of best detection architectures, we connect darknet base YOLO detection pipeline with JDE tracker. As Faster R-CNN [5] consumes more resources and results in low FPS, so single stage detector can help in increasing tracking speed. For accuracy we already achieved high performance real-time object detection, as discussed in detail in **Section 4.2**. Our final experiment contains tracker evaluation of MC-JDE and Deep Sort trackers on Visdrone MOT Test-Dev benchmark which contains 17 different videos acquired from drone mounted cameras. The dataset contains very challenging video environment which contains day and night scenarios where drone is hovering over the city. Currently, our models are trained on 10 different classes which contain person, pedestrian, car, van, motorbike, bicycle, tricycle, awning-tricycle, bus, and truck. The tracking results are measured on 5 classes, which is the subset of these 10 classes, including **car, bus, truck, pedestrian, and van, as considered by official VisDrone MOT 2021 challenge** [74]. According to a recent report on the VisDrone-MOT-2021 challenge, 29 different approaches were submitted, and eight of them were considered in the article. These innovative strategies in object detection, tracking, and reidentification are described in the tabular report shown in the **Figure 4.4** based on Visdrone MOT benchmarking results.

Algorithm	AP	$AP_{0.25}$	$AP_{0.5}$	$AP_{0.75}$	AP_{car}	AP_{bus}	AP_{trk}	AP_{ped}	AP_{van}
SOMOT	58.61	70.75	61.26	43.84	69.18	63.46	48.45	55.64	56.34
GIAOTracker-Fusion	54.18	63.41	55.35	43.78	69.33	51.05	43.20	55.06	52.26
MMDS	52.68	62.92	53.42	41.69	70.20	40.68	51.94	50.27	50.29
Deep IoU Tracker	48.54	63.16	48.11	34.33	51.97	60.05	37.66	37.06	55.94
Yolo-Deepsort -VisDrone	46.70	57.43	48.92	33.75	60.32	43.61	36.22	40.73	52.62
CenterPointCF	44.03	56.91	44.09	31.09	65.65	39.08	41.47	28.34	45.61
MIYoT	39.35	50.72	39.25	28.10	62.05	30.95	36.10	29.79	37.88
HNet	24.71	33.88	24.35	15.89	56.78	11.90	10.99	27.35	16.50

Figure 4.4: Results evaluated on VisDrone-MOT2021 Challenge's (MOT) data. Each assessment mode's top three findings are bolded and highlighted. Red, green, and blue are used as accent colors.[74]

Metrics used in the evaluation are FPS and MOTA where **MOTA** (Multi-Object Tracking Accuracy) is a summary of total tracking accuracy in terms of false positives, false negatives, and identity switches. FPS is the processing time of frames in one

second. Results in **Figure 4.4** are evaluated on the Visdrone MOT test challenge. We evaluated our MOT trackers on the Visdrone MOT test-dev dataset for 5 classes having 17 different video sequences. As our experiments contain both trackers based on detection network, we didn't separately train our trackers on the MOT dataset. This means the tracker can perform even better after learning on MOT training sequences. **Table 4.5** compares the FPS and MOTA accuracy metric of the proposed MC-JDE Real-time tracker. Meanwhile, in the table, FPS is measured on embedded GPU Jetson AGX Xavier and is presented as range and not in the absolute numbers. This is because for each frame both trackers track targets by matching feature embeddings and distance measurement. So, if the current frame contains a high density of detected objects the tracker has big measuring space to assign IDs to the targets which result in low FPS and ID switches due to heavy occlusions. **Table 4.5** demonstrates the comparison of both MC-JDE and Deep Sort trackers. In the table its seen that MC-JDE performs better in terms of both speed and accuracy. The reason is that MC-JDE generates embedding at three different scales from YOLOv3 [10] network model, which has already proven itself as an object detector. Secondly, prime triplet loss from the JDE tracker [41], used for measuring distance among these embeddings also differentiates them well from each other to assign correct IDs. In **Table 4.5**, MOT Accuracy (MOTA) for all 5 classes and their FPS have resulted for both trackers. **Table 4.6** shows comparison with other trackers. Though MC-JDE average AP is low but for class Van, Bus and Truck it has high score then MAD [76] and deepsort_v2 [75] [40] algorithm. Average AP of MC-MOT can be improved after learning on MOT dataset to stand among the other MOT algorithms.

Table 4.5.: FPS and accuracy comparison among Deep Sort and MC-JDE tracker obtained from VisDrone 2018 MOT toolkit evaluated on VisDrone MOT Test-dev dataset.

Method: YOLOv3	MOTA	<i>MOTP_{All}</i>	FPS Range
MC-JDE	20.2	72.2	(3 ~ 14)
Deep Sort	10.8	72.1	(1~ 3)

Table 4.6: Comparison among other tracker on VisDrone MOT Test-Dev data. (Note: The Deep Sort and MC-JDE tracker are not trained on MOT dataset). Bold figures show better results of our trackers without training on MOT data. [75].

Algorithm	AP	$AP_{0.25}$	AP_{Ped}	AP_{Car}	AP_{Van}	AP_{Bus}	AP_{Truck}
YOLOv3							
C-Track [77]	16.12	22.40	7.95	27.74	8.31	28.45	8.15
Deepsort_d2	10.47	17.26	7.12	29.14	10.25	2.38	3.46
MAD	7.27	12.72	7.12	29.14	1.46	1.65	2.85
MC-JDE	4.73	8.85	1.45	4.23	7.97	5.07	4.91
Deep Sort	4.49	8.37	1.38	11.36	3.54	1.17	5.02

CHAPTER 5

5. CONCLUSIONS AND FUTURE WORK

In this thesis we try to tackle the two main problems for multi object detection and tracking for Wide Area Surveillance (WAS) applications: Developing an accurate detector that will enhance tracking pipeline and then maintaining this pipeline to work in real-time, which further requires transforming existing models to light weight form, where they can operate accurately in real-time. To detect objects in wide area imagery, the first task is to design an object detection architecture for small objects. For this purpose, we modify YOLOv4 architecture by interconnecting up-sampling layers and re-train the architecture on Visdrone Object Detection dataset [2]. This results in more robust and accurate features for small objects and this modification increases mAP by 2% when compared to the original YOLOv4 model with no effect on inference speed. As our second task we consider improving FPS by applying CPU and GPU based optimization techniques. For CPU, we propose multi-threading pipeline to divide the load on three threads instead of single thread. For GPU based optimization we convert existing trained models to mixed precision format like FP32 and FP16 using TensorRT pipeline. Both techniques lead to increase in FPS, but at the same time TensorRT pipeline drops some accuracy while converting to low precision formats. So, to regain accuracy, we must fine-tune these models, which is not easy to apply on TensorRT models. So as our third contribution we perform Parameters Reduction and Network Slimming on our base models under different experimental conditions. This results in temporary loss of mAP; but after fine-tuning we regain a level of accuracy which is sometimes even better than the base model. Finally, as our last contribution, we perform data augmentation on Visdrone dataset by applying tiling and then zooming on existing images, which also leads to improvement in mAP as compared to training on original Visdrone Detection dataset. After all, we embed our best detector in Multi Object Tracking (MOT) pipeline to get best results in the term of speed and accuracy.

The current proposed MOT pipeline has still some limitations in maintaining speed in real-time and higher accuracy especially in drone surveillance. This is because that area covered by the drone is much wide, containing detected objects at very high density. First, it's a big challenge for object detection to acquire all desired small targets, even if the object detector performs well, the appearance descriptor or embeddings for these small objects are not discriminative enough to match them successfully in the upcoming frames. Finally, the higher number of targets creates the complex graph at the time of matching association; so, if the object density is high in a particular frame, then the association matrix will be more complex and will affect the real-time performance of the system. Moreover, targets overlapping each other also results in ID switches which leads to accuracy drop. We also experience that the proposed detection-based object tracker can just extract visual and spatial-temporal information and thus is limited to the well-illuminated scenario. So as future work, there should be some deep learning-based motion model to extract more robust features for each detected object. This might work well in UAVs scenarios, as small objects from high altitudes have low-speed motion trajectories. The trajectory estimation can be added as a learning objective of deep network and the network can learn to predict future state as well, especially in both bright and dark scenes. To optimize the real-time performance of the tracker, the generated feature embeddings can be processed in lower precision or using special hashing techniques. These issues are open for future research.

BIBLIOGRAPHY

- [1] Nvidia TensorRT Release Notes, Nvidia Corporation, <https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/index.html>, 2021. (Accessed on Dec 19, 2021).
- [2] P. Zhu *et al.*, “Vision Meets Drones: Past, Present and Future”, arXiv preprint arXiv:2001.06303, 2020.
- [3] L. Liu and M. Tamer Özsu, “Mean average precision,” in *Encyclopedia of Database Systems*, Springer, 2009, doi:10.1007/978-0-387-39940-9_3032.
- [4] W. Liu *et al.*, “SSD: Single Shot multibox Detector,” In *European Conference on Computer Vision (ICCV)* 2018, pp. 21-37.
- [5] S. Ren, K. He, R. Girshick and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137-1149, doi: 10.1109/TPAMI.2016.2577031.
- [6] He. Kaiming *et al.*, “Mask R-CNN,” in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 2980-2988, doi: 10.1109/ICCV.2017.322.
- [7] J. Redmon and A. Farhadi, "YOLO9000: Better, Faster, Stronger," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 6517-6525, doi: 10.1109/CVPR.2017.690.
- [8] A. Sherstinsky, “Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network,” in *Phys. D: Nonlinear Phenomena*, vol. 404, 2020.
- [9] S. Hochreiter and J. Schmidhuber, "Long short-term memory," in *Neural computation*, vol. 9, no. 8, pp. 1735-1780, 1997.
- [10] J. Redmon and A. Farhadi, YOLOv3: An Incremental Improvement, 2018.
- [11] A. Bochkovskiy *et al.*, YOLOv4: Optimal Speed and Accuracy of Object Detection, 2020.
- [12] H. Hu *et al.*, “Joint Monocular 3D Vehicle Detection and Tracking,” in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 5389-5398, doi: 10.1109/ICCV.2019.00549.
- [13] A. Geiger, P. Lenz, C. Stiller and R. Urtasun, “Vision meets robotics: The KITTI dataset”, in *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231-1237, 2013.
- [14] MF. Chang *et al.*, “Argoverse: 3D tracking and forecasting with rich maps”, in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, pp. 8740-8749, 2019.
- [15] R. Girshick, “Fast R-CNN,” in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1440-1448, doi: 10.1109/ICCV.2015.169.
- [16] M. Puterman, *Markov Decision Processes-Discrete Stochastic Dynamic Programming*, New York: John Wiley & Sons, Inc., 1994.

- [17] M. Jiang, C. Deng, Z. Pan, L. Wang and X. Sun, “Multiobject Tracking in Videos Based on LSTM and Deep Reinforcement Learning”, *Complexity*, Nov. 2018.
- [18] T. Fernando, S. Denman, S. Sridharan and C. Fookes, “Tracking by Prediction: A Deep Generative Model for Mutli-person Localisation and Tracking,” in *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2018, pp. 1122-1132, doi: 10.1109/WACV.2018.00128.
- [19] M. Andriluka, S. Roth and B. Schiele, “Monocular 3D pose estimation and tracking by detection,” in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), 2010*, pp. 623-630, doi: 10.1109/CVPR.2010.5540156.
- [20] S. Liu and W. Deng, “Very deep convolutional neural network-based image classification using small training sample size,” in *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*, 2015, pp. 730-734, doi: 10.1109/ACPR.2015.7486599.
- [21] MATLAB, 2020. *version 7.10.0 (R2020a)*, Natick, Massachusetts: The MathWorks Inc, 2020.
- [22] Y. Hu, M. Xiao, K. Zhang, and X. Wang, “Aerial Infrared Target Tracking in Complex Background Based on Combined Tracking and Detecting”, in *Mathematical Problems in Engineering*, 2019, 1-17. 10.1155/2019/2419579.
- [23] C. Ma, X. Yang, C. Zhang, and M. -H. Yang, “Long-term correlation tracking,” in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 5388-5396, doi: 10.1109/CVPR.2015.7299177.
- [24] W. Ou, D. Yuan, Di, Q. Liu, Y. Cao, “Object tracking based on online representative sample selection via non-negative least square”, *Multimedia Tools and Applications*, 2018, vol. 77, pp. 10569-10587, May 2018.
- [25] K. Lebeda *et al.*, “The Thermal Infrared Visual Object Tracking VOT-TIR”, in *2016 Challenge Results*, 2016, 9914. 824-849. 10.1007/978-3-319-48881-3_55.
- [26] C. Wei and S. Jiang, “Automatic Target Detection and Tracking in FLIR Image Sequences Using Morphological Connected Operator,” in *2008 International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, 2008, pp. 414-417, doi: 10.1109/IIH-MSP.2008.193.
- [27] M. Abadi, A. Agarwal *et al.*, “Tensorflow: Large-scale machine learning on heterogeneous distributed systems”, 2016.
- [28] G. Ning *et al.*, “Spatially supervised recurrent convolutional neural networks for visual object tracking,” in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2017, pp. 1-4, doi: 10.1109/ISCAS.2017.8050867.
- [29] S. Yun and S. Kim, “Recurrent YOLO and LSTM-based IR single pedestrian tracking,” in *19th International Conference on Control, Automation and Systems (ICCAS)*, 2019, pp. 94-96, doi: 10.23919/ICCAS47443.2019.8971679.

- [30] J. Deng *et al.*, “ImageNet: A large-scale hierarchical image database,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009, pp. 248-255, doi: 10.1109/CVPR.2009.5206848.
- [31] C. Cortes & V. Vapnik, Support-vector networks. *Machine learning*, 1995, 20(3), pp.273–297.
- [32] M. Everingham, A. Zisserman, C.K.I. Williams and L. Van Gool, “The PASCAL Visual Object Classes Challenge 2006 (VOC 2006) Results”, 2006.
- [33] A. Basharat *et al.*, “Real-time multi-target tracking at 210 megapixels/second in Wide Area Motion Imagery,” in *IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2014, pp. 839-846, doi: 10.1109/WACV.2014.6836016.
- [34] T. Rovito, *et al.*, “Columbus Large Image Format (CLIF) 2007 Dataset”, 2018.
- [35] C. Cohenour, F. van Graas, R. Price and T. Rovito, “Camera models for the wright patterson air force base (WPAFB) 2009 wide-area motion imagery (WAMI) data set,” in *IEEE Aerospace and Electronic Systems Magazine*, vol. 30, no. 6, pp. 4-15, June 2015, doi: 10.1109/MAES.2015.140150.
- [36] J. Bowman, S. Emerson, and M. Darnovsky, “The Practical SQL Handbook: Using SQL Variants”, in *Addison-Wesley Professional*, vol.4, 2001.
- [37] K. Fieldhouse *et al.*, “KWIVER: An open-source cross-platform video exploitation framework,” in *2014 IEEE Applied Imagery Pattern Recognition Workshop (AIPR)*, 2014, pp. 1-4, doi: 10.1109/AIPR.2014.7041910.
- [38] P. Ghaemmaghami, “Tracking of Humans in Video Stream Using LSTM Recurrent Neural Network.”, 2017.
- [39] A. S. Mohammad, S. Saleem, A. Dilawari and M. U. Ghani Khan, “Cross Input Neighborhood Difference for Re-identification of Occupational Humans,” in *22nd International Multitopic Conference (INMIC)*, 2019, pp. 1-6, doi: 10.1109/INMIC48123.2019.9022783.
- [40] X. Hou, Y. Wang and L. Chau, “Vehicle Tracking Using Deep SORT with Low Confidence Track Filtering,” in *16th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, 2019, pp. 1-6, doi: 10.1109/AVSS.2019.8909903.
- [41] H. Kieritz, W. Hübner and M. Arens, “Joint Detection and Online Multi-object Tracking,” in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2018, pp. 1540-15408, doi: 10.1109/CVPRW.2018.00195.
- [42] T. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan and S. Belongie, “Feature Pyramid Networks for Object Detection,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 936-944, doi: 10.1109/CVPR.2017.106, 2017.
- [43] R. Girshick, J. Donahue, T. Darrell and J. Malik, “Region-based convolutional networks for accurate object detection and segmentation”, *TPAMI*, 2015.
- [44] R. Girshick, “Fast R-CNN,” in *IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1440-1448, doi: 10.1109/ICCV.2015.169.

- [45] P. J. Hargrave, "A tutorial introduction to Kalman filtering," *IEE Colloquium on Kalman Filters: Introduction, Applications and Future Developments*, 1989, pp. 1/1-1/6.
- [46] A Ammar, A Koubaa, M Ahmed, A Saad and B Benjdira, "Vehicle Detection from Aerial Images Using Deep Learning: A Comparative Study", in *Electronics 10*, 2021.
- [47] C. Wang *et al.*, "CSPNet: A New Backbone that can Enhance Learning Capability of CNN" *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2020: 1571-1580.
- [48] S. Xie, R. Girshick, P. Dollár, Z. Tu and K. He, "Aggregated Residual Transformations for Deep Neural Networks," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 5987-5995, doi: 10.1109/CVPR.2017.634.
- [49] J. Redmon, Darknet: Open-Source Neural Networks in C, <http://pjreddie.com/darknet>, 2013-2016. (Accessed on Nov 10, 2021).
- [50] M. Tan and Q. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks" in *36th International Conference on Machine Learning*, pp. 6105-6114, 2019.
- [51] J. Yang, X. Fu, Y. Hu, Y. Huang, X. Ding and J. Paisley, "PanNet: A Deep Network Architecture for Pan-Sharpening," in *IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 1753-1761, doi: 10.1109/ICCV.2017.193.
- [52] T.-Y. Lin *et al.*, "Feature Pyramid Networks for Object Detection." in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017: 936-944.
- [53] G. Huang *et al.*, "Densely Connected Convolutional Networks." in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, 2261-2269.
- [54] K. He, X. Zhang, S. Ren and J. Sun, "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 9, pp. 1904-1916, 1 Sept. 2015, doi: 10.1109/TPAMI.2015.2389824.
- [55] A. F. Agarap *et al.*, "Deep learning using rectified linear units (ReLU)", *arXiv:1803.08375*, 2018.
- [56] D. Mishra *et al.*, Mish: "A Self Regularized Non-Monotonic Neural Activation Function." *ArXiv, abs/1908.08681*, 2019
- [57] P. Ramachandran, B. Zoph and V. Quoc, "Swish: A Self-Gated Activation Function", 2017.
- [58] G. Ghiasi *et al.*, "DropBlock: A regularization method for convolutional networks.", in *NeurIPS*, 2018.
- [59] T.-Y. Lin *et al.*, "Microsoft COCO: Common Objects in Context", in *13th European Conference of Computer Vision (ECCV)*, pp. 740-755, 2014.
- [60] S. Ali, A. Siddique, H. F. Ateş and B. K. Güntürk, "Improved YOLOv4 for Aerial Object Detection," in *29th Signal Processing and Communications*

- Applications Conference (SIU)*, 2021, pp. 1-4, doi: 10.1109/SIU53274.2021.9478027.
- [61] P. Zhu *et al.*, “VisDrone-VDT2018: The Vision Meets Drone Video Detection and Tracking Challenge Results”, 2019, 10.1007/978-3-030-11021-5_29.
- [62] A. Paszke, *et al.*, “Pytorch: An Imperative Style High-Performance Deep Learning Library,” in *Neural Information Processing Systems*, vol. 32, pp. 8024-8035, 2019.
- [63] F. Bastien *et al.*, “Theano: New Features and Speed Improvements”, in *NIPS*, 2012.
- [64] T. Hoefler *et al.*, “Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks”, in *Tech Report*, Feb. 2021.
- [65] Y. Jia *et al.*, “Caffe: Convolutional Architecture for Fast Feature Embedding”, *ACM International Conference on Multimedia*, pp. 675-678, 2014.
- [66] T. Chen *et al.*, “MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems.”, 2015.
- [67] S. Srinivas, A. Subramanya and R. V. Babu, “Training Sparse Neural Networks,” in *IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2017, pp. 455-462, doi: 10.1109/CVPRW.2017.61.
- [68] S. Ioffe and C. Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.”, 2015.
- [69] P. Zhang, Y. Zhong and X. Li, “SlimYOLOv3: Narrower, Faster and Better for Real-Time UAV Applications,” in *IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, 2019, pp. 37-45, doi: 10.1109/ICCVW.2019.00011.
- [70] N. Wojke, A. Bewley and D. Paulus, “Simple online and real-time tracking with a deep association metric,” in *IEEE International Conference on Image Processing (ICIP)*, 2017, pp. 3645-3649, doi: 10.1109/ICIP.2017.8296962.
- [71] Y. Dodge, “Mahalanobis Distance,” in *The Concise Encyclopedia of Statistics*, Springer, New York, NY. https://doi.org/10.1007/978-0-387-32833-1_240.
- [72] G. Sicuro, “The Euclidean Matching Problem”, Cham, Switzerland: Springer, 2017
- [73] F. Rahutomo, T. Kitasuka, and M. Aritsugi, “Semantic Cosine Similarity”, 2012.
- [74] G. Chen *et al.*, “VisDrone-MOT2021: The Vision Meets Drone Multiple Object Tracking Challenge Results,” in *IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, 2021, pp. 2839-2846, doi: 10.1109/ICCVW54120.2021.00318.
- [75] P. Zhu *et al.*, “VisDrone-VDT2018: The Vision Meets Drone Video Detection and Tracking Challenge Results,” in *Leal-Taixé L., Roth S. (eds) Computer Vision – ECCV 2018 Workshops*, 2018

- [76] J. Valmadre *et al.*, “End-To End Representation Learning for Correlation Filter-Based Tracking,” in *Proceedings of IEEE Conference on Computer Vision and Pattern*, 2017, pp. 5000–5008.
- [77] W. Tian and M. Lauer, “Joint tracking with Event Grouping and Temporal Constraints,” in *IEEE International Conference on Advanced Video and Signal-Based Surveillance*, 2017, pp. 1–5.

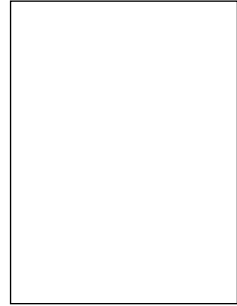


CURRICULUM VITAE

Name Surname :

Place and Date of Birth :

E-Mail :



EDUCATION:

B.Sc. : 2019, COMSATS University, Faculty of Computer Sciences
CS Department.

PROFESSIONAL EXPERIENCE AND REWARDS:

- 2019, Best Desktop Application award at Project Exhibition, COMSATS University Islamabad
- 2015-2019, Full Scholarship Awardee Student, COMSATS University

PUBLICATIONS, PRESENTATIONS, AND PATENTS ON THE THESIS:

- S. Ali, A. Siddique, H. F. Ateş and B. K. Güntürk, "Improved YOLOv4 for Aerial Object Detection," *2021 29th Signal Processing and Communications Applications Conference (SIU)*, 2021, pp. 1-4, doi: 10.1109/SIU53274.2021.947802.

ii DEEP LEARNING MODEL OPTIMIZATIONS FOR REAL-TIME SMALL OBJECT DETECTION ON EMBEDDED GPUS

ORIGINALITY REPORT

8%

SIMILARITY INDEX

4%

INTERNET SOURCES

7%

PUBLICATIONS

2%

STUDENT PAPERS

PRIMARY SOURCES

1	arxiv.org Internet Source	2%
2	Submitted to Institute of Technology Carlow Student Paper	1%
3	Guanghan Ning, Zhi Zhang, Chen Huang, Xiaobo Ren, Haohong Wang, Canhui Cai, Zhihai He. "Spatially supervised recurrent convolutional neural networks for visual object tracking", 2017 IEEE International Symposium on Circuits and Systems (ISCAS), 2017 Publication	<1%
4	Ming-xin Jiang, Chao Deng, Zhi-geng Pan, Lanfang Wang, Xing Sun. "Multiobject Tracking in Videos Based on LSTM and Deep Reinforcement Learning", Complexity, 2018 Publication	<1%
5	www.hindawi.com Internet Source	<1%
