7-2022

Smart and Intelligent Automation for Industry 4.0 using Millimeter-Wave and Deep Reinforcement Learning

Abhishek Vashist
av8911@rit.edu

Smart and Intelligent Automation for Industry 4.0 using Millimeter-Wave and Deep Reinforcement Learning

Abhishek Vashist

Smart and Intelligent Automation for Industry 4.0 using Millimeter-Wave and Deep Reinforcement Learning

Abhishek Vashist

July, 2022

A Dissertation Submitted
in Partial Fulfillment
of the Requirements for the Degree of
Doctorate of Philosophy
in
Engineering

RIT | Kate Gleason College of Engineering

Smart and Intelligent Automation for Industry 4.0 using Millimeter-Wave and Deep Reinforcement Learning

Abhishek Vashist

Committee Approval:

Dr. Amlan Ganguly *Advisor* Date
Department of Computer Engineering

Dr. Andres Kwasinski Date
Department of Computer Engineering

Dr. Michael E. Kuhl Date
Department of Industrial and Systems Engineering

Dr. Clark Hochgraf Date
Department of Electrical, Computer, and Telecommunications Engineering Technology

Acknowledgments

I would like to express my sincere appreciation and gratitude to my advisor Dr. Amlan Ganguly for his constructive remarks, optimal guidance, devoted time, and contributions throughout my Ph.D. With his guidance and belief in my abilities, I can successfully finish my Ph.D. journey. He always welcomed my crazy ideas and provided me with the right advice.

I would also like to acknowledge Dr. Sai Manoj PD, George Mason University, VA, for being an amazing collaborator with all the projects that I have been a part of during my Ph.D.

I am also grateful to my committee members Dr. Michael E. Kuhl, Dr. Andres Kwasinski, and Dr. Clark Hochgraf, who supported me in my efforts and provided me with constructive feedback to enhance the quality of my work.

I want to thank my lab mates M. Meraj Ahmed, Sayed Ashraf Mamun, Andrew Keats, Purab Ranjan, and Rahul Singh Gulia, who always helped me with my research efforts and always provided me with a great collaborating environment.

Finally, I want to thank my friends Robert Relyea, Piers Kwan, and Andrew Meyer for all the fun we had during our time at RIT, especially the coffee runs.

To science and the hope for a better tomorrow.

Abstract

Innovations in communication systems, compute hardware, and deep learning algorithms have led to the advancement of smart industry automation. Smart automation includes industrial sectors such as intelligent warehouse management, smart infrastructure for first responders, and smart monitoring systems. Automation aims to maximize efficiency, safety, and reliability. Autonomous forklifts can significantly increase productivity, reduce safety-related accidents, and improve operation speed to enhance the efficiency of a warehouse. Forklifts or robotic agents are required to perform different tasks such as position estimation, mapping, and dispatching. Each of the tasks involves different requirements and design constraints. Smart infrastructure for first responder applications requires robotic agents like Unmanned Aerial Vehicles (UAVs) to provide situation awareness surrounding an emergency. An immediate and efficient response to a safety-critical situation is crucial, as a better first response significantly impacts the safety and recovery of parties involved. But these UAVs lack the computational power required to run Deep Neural Networks (DNNs) that are used to provide the necessary intelligence.

In this dissertation, we focus on two applications in smart industry automation. In the first part, we target smart warehouse automation for Intelligent Material Handling (IMH), where we design an accurate and robust Machine Learning (ML) based indoor localization system for robotic agents working in a warehouse. The localization system utilizes millimeter-wave (mmWave) wireless sensors to provide feature information in the form of a radio map which the ML model uses to learn indoor positioning. In the second part, we target smart infrastructure for first responders, where we present a computationally efficient adaptive exit strategy in multi-exit Deep Neural Networks using Deep Reinforcement Learning (DRL). The proposed adaptive exit strategy provides faster inference time and significantly reduces computations.

Contents

List of Figures

List of Tables

Acronyms

AI	Artificial Intelligence
APs	Access Points
AU	Aggregation Unit
CDF	cumulative distribution function
DQN	Deep Q Network
DRL	Deep Reinforcement Learning
EMS	Emergency Medical Service
IMH	Intelligent Material Handling
IU	Imputation Unit

KF	Kalman Filter
KNN	K Nearest Neighbor
LoS	Line of Sight
LR	Linear Regression
MAC	Multiply and Accumulate
MAE	Mean Absolute Error
ML	Machine Learning
mmWave	Millimeter Wave
NAS	Network Architecture Search
SDU	Synthetic Data Unit

SNR

Signal to Noise Ratio

SVM

Support Vector Machine

SVR

Support Vector Regression

UAVs

Unmanned Aerial Vehicles

Chapter 1

Introduction

1.1 Smart Industry Automation

With the advent of industry 4.0, many industrial sectors are developing and deploying new intelligent and smart technologies [6–8]. Studies have shown a significant increase in investment to further advance the progress in smart automation. A report published by Statista [9] projected smart industries to have a market size of over $1.5 trillion by 2025. Companies are investing significantly in smart warehouse management, smart manufacturing, autonomous vehicles, and smart power grids. With such technological innovations, sectors such as warehouse management benefit from an increase in worker productivity and a significant increase in worker safety. Occupational Safety and Health Administration (OSHA) reported approximately 35,000 severe injuries due to forklift accidents each year [10].

Smart automation combines various technologies, including intelligent software systems, communication infrastructures, and high-performance computation hardware. Smart automation comprises of many different application space including Internet of Things (IoT) [11,12], Cyber Physical Systems (CPS) [13,14], smart manufacturing [15], intelligent warehouse management [16,17], and smart cities [6,18]. Each of them requires different technological innovations based on the requirement and constraints. Performing various tasks involved in smart automation requires

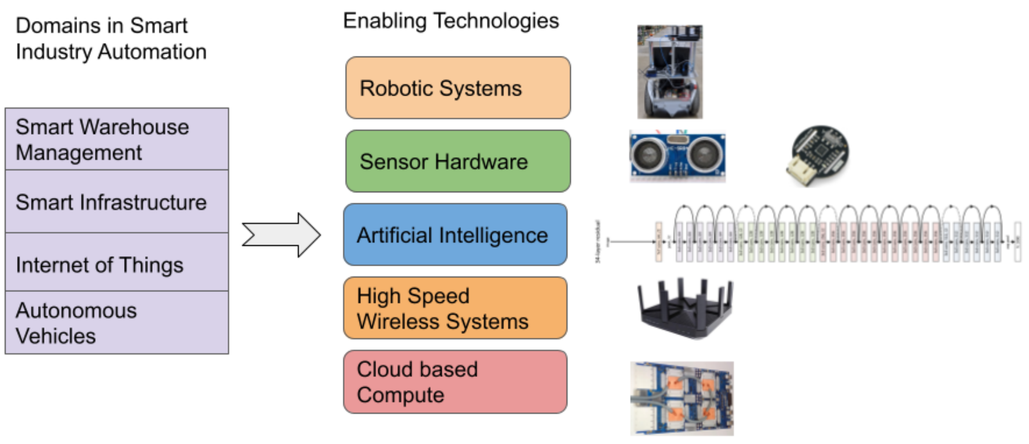


Figure 1.1: Different domains and enabling technologies in Smart Industry automation

information such as scene classification for drone-based applications [19–21], object recognition and localization in autonomous agents [22–25], and intelligent dispatch and scheduling in smart warehouse management [26, 27]. While the specific information may change based on the application, the technological innovations enabling the growth of industrial automation remain common across many of these industrial sectors. Figure 1.1 illustrates different domains and the technologies required to enable such innovations.

In particular, technologies that have played dominant role in widespread adoption of industry automation are robotics [8], deep learning [28], and high speed wireless systems [29]. Deep neural networks (DNNs) are state-of-the-art for applications that require extracting information such as object detection, position estimation, and scene classification. Advancements in robotics have enabled the development of various critical components that provide autonomy in sectors like intelligent material handling (IMH) and smart manufacturing [15]. The communication system forms the backbone for sharing and collaborating information and data among multiple agents working in an autonomous environment. Sharing of high-fidelity of information can quickly saturate the communication links. With recent advancements in millimeter-wave technology, the wireless links can now provide multi-gigabits of data bandwidth [30].

In this work, we have considered two major application domains in smart industry automation, warehouse management for intelligent material handling (IMH) and smart infrastructure for first responders. We consider smart warehouse management as our application domain in the first part. Smart and automated warehouses have been a critical focus in industrial automation. Automation in warehouses consists of advanced communication systems, autonomous navigation, and automation of machines [31–33]. Precise and robust localization is one of the critical requirements as it forms the basis for navigating autonomous agents. For indoor localization, wireless sensor-based approaches are vastly investigated [34] and have been the preferred positioning approach for indoor environments due to the low cost, easy deployment, and power efficiency. Here, we address the challenge of providing a robust and highly accurate positioning system for autonomous agents in an indoor environment by designing a machine learning-based (ML) localization system. The proposed ML localization system uses wireless features from 60 GHz millimeter-wave communication routers.

In the second part, we consider smart infrastructure for public safety applications. Applications such as public safety, inspection, and monitoring require using sensor networks working on mobile or edge devices to provide the relevant information to the agency. Low compute mobile or local edge devices are often used in such applications. These applications use UAV-based mobile devices for performing various tasks, including situation awareness for first responders [20] and monitoring infrastructure for buildings [21] and agricultural crops [19]. The challenges and the constraints associated with these applications are the low compute capability on mobile devices that run computationally heavy DNNs. To address the challenges and provide the required intelligence on low compute devices, we present the design and implementation of multi-exit DNNs with a Deep Q Network-based adaptive exit strategy. The proposed adaptive exit strategy provides faster inference speed and low energy consumption.

1.2 Thesis Contribution

The major contributions of this dissertation are summarized below:

- **Machine learning and kalman filter integrated wireless localization:**
  - We present design of an indoor warehouse localization system using consumer grade off-the-shelf 60 GHz wireless routers. For this, we use Signal-to-Noise-Ratio (SNR) as a feature from consumer-grade wireless Access Points (APs) in Machine Learning based localization algorithm.
  - We present an approach to deal with the missing wireless feature information from the APs during the training and inference of the ML models. The consumer-grade APs can lose connectivity intermittently, which can cause severe performance degradation. This approach to imputation is made by modeling the SNR characteristics using the collected dataset and further augmenting it with synthetic data.
- **DQN based adaptive exit selection in multi-exit DNNs:**
  - We propose a DQN based adaptive exit selection in multi-exit DNNs to optimize exit selection in multi-exit DNNs. The DQN network is first trained to learn an exit policy utilizing history of state information. The state information includes accuracy, inference latency, previous exit decision, and image complexity. History of recent frames is used to exploit the temporal and spatial correlation in input images, resulting in a more accurate and robust exit selection.
  - We implement an image complexity approach to quantify the complexity associated with an input image. The complexity is computed based on the minimum number of points required to represent all contours present in

an input image. Utilizing complexity the DQN provides a more adaptive exit strategy for sequentially time and space dependent frames.

1.3 Dissertation Organization

This Dissertation is organized into five chapters. Brief information about each chapter is mentioned below:

- **Introduction** Here, we introduce the motivation behind the work that has been performed toward completing this thesis. Then, it states the contribution of this dissertation and dissertation organization.

- **Background and Related Work** In this chapter, we explain the background and review the related work for wireless localization, multi-exit DNNs, and deep reinforcement learning.

- **Millimeter-wave Indoor Localization using Machine Learning** In this chapter, we describe the different components involved in the proposed localization model along with the detailed experimental evaluation in a warehouse environment.

- **Adaptive Exit Selection using DQN in Multi-Exit Networks** In this chapter, we discuss the design and architecture of proposed adaptive exit strategy in multi-exit DNNs. Then we provide a detailed experimental evaluation of the proposed system using three different classes of multi-exit DNNs.

- **Future Work :** Here we describe the future directions the work can be extended.

Chapter 2

Background and Related Work

In this chapter, we discuss the background and review the related work in indoor wireless localization, multi-exit deep neural networks, and deep reinforcement learning.

2.1 Wireless Localization

Different wireless sensors such as Wi-Fi [35], Bluetooth [36], and RFID [37] have been investigated for indoor localization. But, due to higher coverage, ease of deployment, and ability to provide communication and localization simultaneously, Wi-Fi-based approaches are more popular. Millimeter-wave frequencies ranging between 30 GHz to 300 GHz, particularly in the unlicensed 60 GHz spectrum, allow higher data rates of multi-gigabit-per-second, making them suitable for many applications requiring high-speed wireless data rates such as robot navigation [8], smart cities [6], and medical applications [38]. Hence, for such applications, it is suitable to utilize the mmWave technology for providing the communication infrastructure.

For localization in indoor environments such as warehouses, wireless approaches are vastly investigated [34,35,39,40] and have been the preferred positioning approach for the indoor environments due to the low cost, easy deployment, and power efficiency. Wireless sensor-based localization techniques can be broadly classified into two technologies. The first technique uses the channel propagation model to esti-

Table 2.1: Performance comparison with different sensor based localization techniques

Sensor	Accuracy	Advantages	Disadvantages
GPS	Around 10m	Low cost sensor	Poor performance in indoor environment
Camera [43]	Mean error of 0.75m	Low cost sensor	Needs well illuminated environment
LiDAR SLAM [44]	0.07m to 0.03m RMSE	High Accuracy	High computation cost

mate the distance to the receiver, also known as the Client, using the wireless signal strength information. Then using the known locations of the Access Points (APs), trilateration is used to predict the Client's location.

In channel model based localization approaches [41], the known wireless channel model is used to estimate the distance between the sender and the receiver node. The sender transmits the signal and suffers from signal degradation. Using the known channel model, we can estimate the distance based on the strength of the signal received at the receiver. The farther the receiver is from the sender, the more degradation will occur, and the lower the received signal strength at the receiver. Wireless channels are vulnerable to shadowing, atmospheric effects, and attenuation due to materials [42]. So, such effects must be considered while estimating the channel model, making channel estimation a non-trivial task.

The second technique estimates the Client's position by matching the known wireless signal strength from the APs. This is done by collecting the signal strength information at many locations within the environment and then using it as a matching database. This technique is known as wireless fingerprinting. In [45], the authors present a moving average based k-Nearest neighbor approach for fingerprint-based wireless localization. The accuracy achieved by their system is low.

Authors in [46, 47] designed support vector regression for localization using the received signal strength indicator (RSSI) fingerprinting as input features. Authors in [48] utilize a Deep Reinforcement Learning (DRL) framework for indoor localization using Bluetooth devices. The idea is to mitigate the data collection step for training the ML model and use reinforcement learning algorithms to learn the location predictions. Their approach results in low accuracy with a Root Mean Square Error (RMSE) of 12.2m. In [49] authors used mmWave routers in an indoor office environment and have shown a high accuracy of 98.8%. But their approach was evaluated in a small-scale environment with seven locations used for the training and testing.

Table 2.1 summarizes the localization techniques using different sensor modalities: Global Positioning System (GPS), Light Detection and Ranging (LiDAR), and vision. Using these sensors, SLAM-based localization approach is applied. LiDAR and vision-based approaches are computationally expensive as the data generated is either a point cloud of all the distances or video streams from hi-fidelity cameras. SLAM approaches typically use a particle filter and an algorithm like Adaptive Monte Carlo Localization AMCL [50] to estimate the robot's position. So, high-performance devices such as Graphics Processing Units (GPUs) are needed on the agent to support localization. Also, adding high-performance hardware becomes a constraint if the agent has a small form factor like UAVs.

2.2 Multi-Exit Deep Neural Networks

DNNs are getting deeper to improve network performance [51–53]. With deeper network architectures, the total number of computations increases, which results in higher inference time. Multi-exit DNNs modify the traditional end-to-end DNNs by introducing additional intermediate classification output layers [2, 3, 54, 55]. Intermediate exits allow the input samples to utilize these exits and reduce the inference time latency and computation cost. The multi-exit approach estimates if the learned feature representation in earlier layers provides sufficient information for the network to make high-accuracy predictions for an input.

Figure 2.1 shows an architecture of multi-exit DNN with multiple intermediate

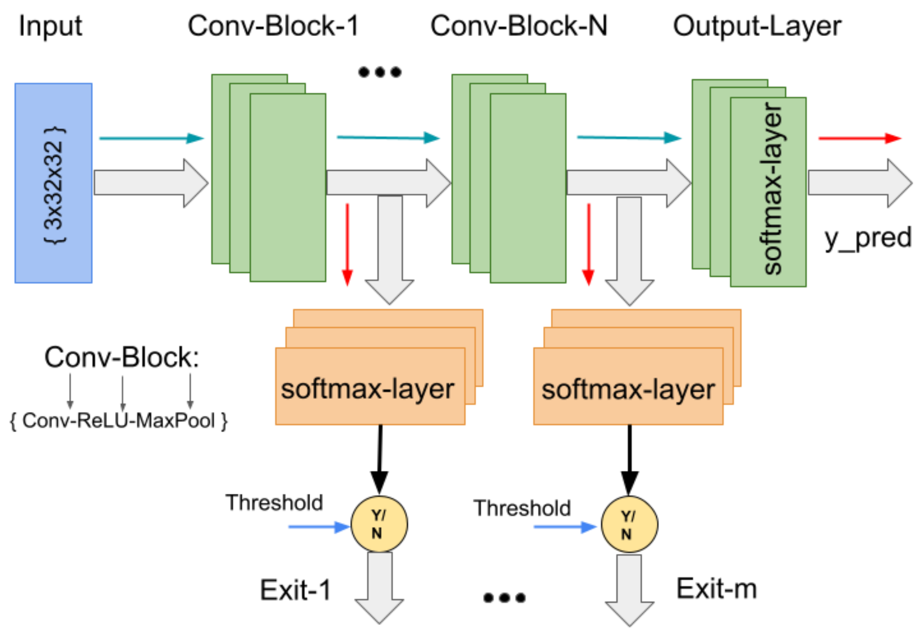


Figure 2.1: Multi-exit DNN with threshold based exit strategy during inference

exits using a threshold-based exit selection. An exit layer can also include additional convolutional and fully connected layers. Whether an exit is selected depends on the exit selection strategy employed. The exit strategy affects the performance that multi-exit DNNs can achieve. The intermediate exits, in a way, represent the decision boundary learned. With earlier exits, the exits learn decision boundaries capable of classifying easier samples correctly but will miss-classify the complex samples, as shown in Fig. 2.2. But with later exits in the network, a more complex decision boundary is learned and capable of correctly classifying hard samples. BranchyNet [2], EENet [54], Early-Bert [3], and SkipNet [55] are current state-of-the-art multi-exit DNN approaches that are discussed next.

BranchyNet: BranchyNet [2] utilizes a entropy-based threshold for exit selection. During training, the loss from all the exits is computed and the final loss is a weighted sum of all the losses. The loss formulation is shown in (2.1).

$$L_{BranchyNet} = \sum_{j=N} w_j L_j \tag{2.1}$$

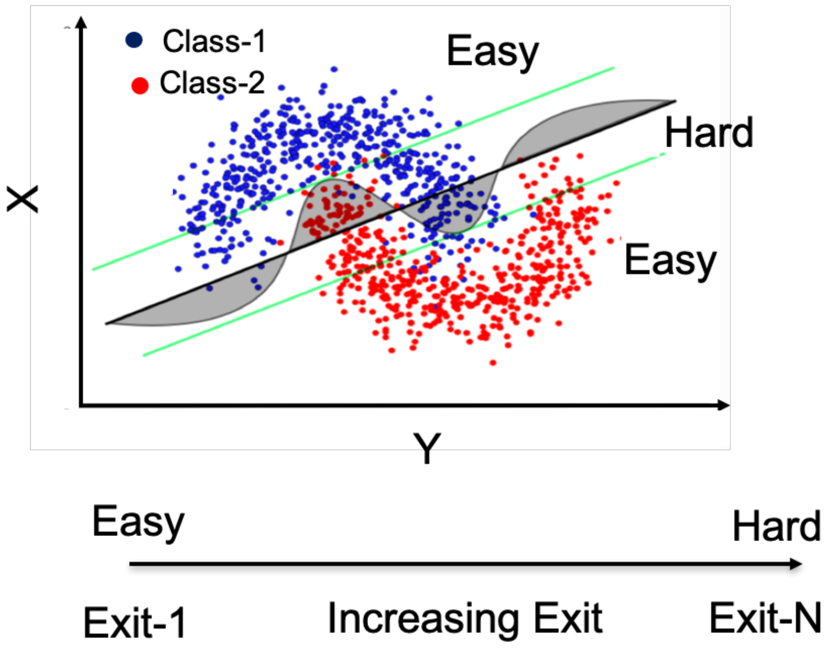


Figure 2.2: Classification of samples with multiple exits in multi-exit DNNs [1]

Where $L_{BranchyNet}$ is the net loss considering all the exit branches in multi-exit DNN. L_j is the loss from input to exit branch j. Further, w_j is the weight associated with different exit branches.

During inference, the network computes the entropy at each exit and compares it with a user-defined threshold. The input sample classifies using the selected exit if the threshold is met. The threshold comparison for each exit happens sequentially, starting from the first exit location to the last classification output layer. Figure 2.3 illustrates the entropy computation at an exit in BranchyNet. The softmax output is used to compute the entropy. The computed entropy value is compared against the selected threshold for exit selection.

EENet: EENet [54] also follows the threshold-based exit selection, where the threshold confidence is measured using a sigmoid function. At each exit, confidence is computed by taking the last fully-connected layer at the output and using another fully-connected layer with a single output node. The output is then passed through a sigmoid function to get the exit confidence. Threshold values are then selected for

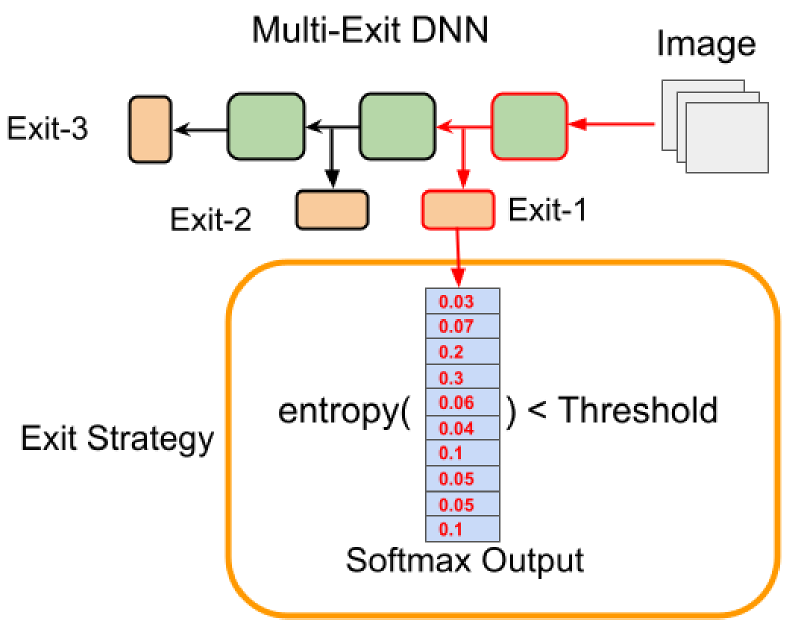


Figure 2.3: Entropy based threshold presented in BranchyNet [2]

exit selection during testing. The approach is similar to BranchyNet, where entropy is used for the exit confidence.

Early Exit Bert: Authors in [3] introduce the idea of a patience-based adaptive early exit in the Bidirectional Encoder Representations from Transformers (BERT) language model. BERT model contains several deep layers and billions of network parameters, making them computationally expensive for many hardware devices. Their approach uses an entropy-based threshold at exits as presented in BranchyNet [2] along with a patience window, where an exit is selected if the prediction at exits is unchanged during the patience window. Figure 2.4 shows the design of such an approach, where an exit is selected if the prediction is unchanged for two successive exits. This work's main contribution is improving network accuracy by reducing the likelihood of miss-predictions at exits. In early exit BERT, the loss function used during the training is the weighted of loss over number of exits, as shown in (2.2) [3].

$$L_{exit-bert} = \frac{\sum_{j=N} L_j}{\sum_{j=N} j} \tag{2.2}$$

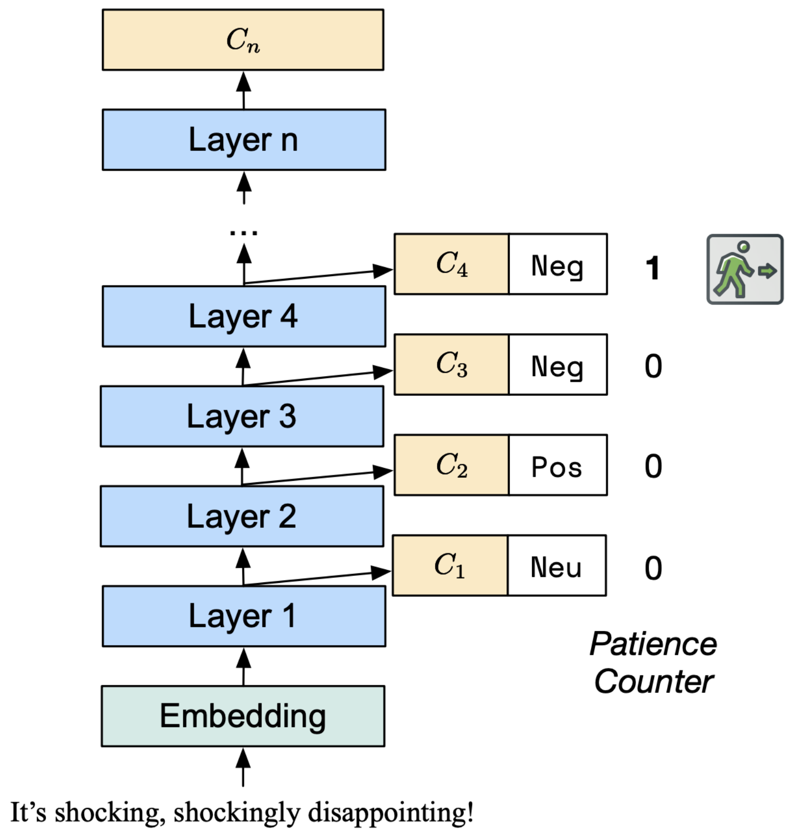


Figure 2.4: Patience based adaptive exit selection in BERT [3]. The exit is only selected if the prediction remain unchanged during the patience window.

SkipNet: SkipNet [55] modifies the residual networks and uses gating layers between the sequential layers. The addition of gating layers makes the skipping decision for the network. The skipping problem is a sequential decision-making process that combines supervised learning and reinforcement learning. The SkipNet [55] uses a gating function to decide if the network should skip or execute the gate. They use the RL approach to learn the task of gate skipping, where the reward is given based on the cost associated with executing the layer, including the gate. The approach is limited to only resent-based architectures, and authors do not show the application to other DNN architectures such as CNNs [51], and MobileNet [52].

Other commonly employed network optimization techniques include weight quantization [56, 57]. While our work focuses on multi-exit DNNs without network quantization, we can utilize quantization techniques as presented in [56, 57] to design the multi-exit DNNs along with adaptive exit selection to further improve the energy efficiency.

2.3 Deep Reinforcement Learning

Supervised machine learning and deep learning algorithms use static dataset and corresponding output labels during the training process. The data provided is independent of each other in time and also referred as independent and identically distributed (iid). Reinforcement learning (RL) is a form of unsupervised learning approach, where the learning algorithm, commonly called as an agent, interacts with the environment, with which it receives the input observations or state (S_t) and performs an action (A_t) and receives a scalar reward value (R_t). Based on the action, the environment produces a new set of state for the agent. The idea being, the agent learns by interacting with the environment and receiving reward values based on the predicted actions, this introduces the notion of a feedback loop in the learning process. The goal of the RL algorithm is to maximize the cumulative received reward during the training process. This is also known as reward hypothesis in reinforcement learning.

The data used during the training is non-iid as the data input in the next time-step will be dependent on the agent's action to the previous time-step input. In traditional Q-table based RL, the learning algorithm uses a table, known as Q-table, to record the mapping between the state and action. Once the algorithm is trained, then the learned Q-table can be used as a look up table to output the desired action based on the state, we use the action and state pair to index the table and select the action for which we expect the maximum Q-value. Where the Q-value represents the expected future reward for the action given the state, mathematically can be written as (2.3). Where, γ is called the discount rate, and the value is less than one. It gives more weight to the rewards which are closer to current time step and diminishing reward value to distant future rewards. It also helps to bound the infinite series sum of to be finite. We use Bellman equation for updating the $Q(s_t, a_t)$ during the learning process as described in (2.4).

$$Q(s_t, a_t) = E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3}...|s_t, a_t] \tag{2.3}$$

$$\hat{Q}(s_t, a_t) = \alpha[R(s, a) + \gamma max Q(\hat{s}, \hat{a}) - Q(s, a)] \tag{2.4}$$

The Q-table based RL algorithm is non-scalable to higher dimensional input space and further suffers if the learning problem is complex in nature. So, instead of using a look-up table, Deep Reinforcement Learning (DRL) utilizes neural networks to approximate the mapping between the states and actions to overcome the issues and challenges of increasing state space that is seen in RL.

DRL has been used in wide variety of research and tasks in the past few years [58] as they not only improve the scalability but have also been proven to be more robust in learning complex behaviors [58–60]. Deep Q Networks (DQNs) are type of DRL where, instead of using a lookup based Q-table, we use a deep neural network (DNN). Then for a given input state, the DNN will approximate the different Q-values for each action. Here, the input is the state and the outputs are the Q-values for different actions.

In DQN, we formulate a loss function and use gradient descent algorithm to update the neural network parameters, as shown in (2.5). We use Temporal Difference (TD) loss in DQN that uses the Q estimates and the Q-Target. Q-Target is estimated using the Bellman equation. The Q-Target is computed as the sum of immediate reward and the discounted maximum Q-value for the next state, this is shown in (2.6). The future rewards computed are considered to be discounted by using a discount factor,

γ.

$$L(\theta) = 1/N \sum_{i=N} (Q(s_t, a_t, \theta) - \hat{Q}(s_t, a_t, \theta))^2 \tag{2.5}$$

$$\hat{Q} = R(s_t, a_t) + \gamma max_{a_t} Q(\hat{s}_t, \hat{a}_t, \theta) \tag{2.6}$$

Additionally, during the training, a replay memory is also used to learn from the previous experience. This helps in reducing the dependence on the current sequential state. With experience replay we store the previously seen information and use it in the agent's learning multiple times. During the training, the DQN agent model learns to select actions that maximizes the cumulative discounted reward by interacting with the environment. The future rewards computed are considered to be discounted by using a discount factor, γ.

Deep reinforcement learning techniques have shown that complex problems can use the experience-based learning and produce state-of-the-art results in wide variety of fields. Authors in [58] designed a DQN algorithm for playing atari games by using a DQN agent with three convolutional layer blocks. The results showed state-of-the-art performance on seven different atari games. Authors in [59] uses the approach of DQN network for Cloud Service Providers (CSP) resource management and achieved high energy efficiency and low runtime with faster convergence. Work in [61] presented a DRL approach for robot motion for autonomous navigation with the ability to move within crowded space with human like speed and performing motions like crossing, passing and overtaking.

Chapter 3

Millimeter-wave Indoor Localization using Machine Learning

3.1 Introduction

An accurate and cost-effective indoor positioning system is one of the major requirements for automation in the next generation industrial revolution, Industry 4.0, and beyond. Smart and automated warehouses have been a key focus in industrial development, which includes advanced communication systems, autonomous navigation, and automation of machines [31–33]. One of the preliminary requirements for the agent is to localize itself within the environment with high accuracy for coordination and carrying out tasks. Localization requires information from various kinds of sensors. Sensors such as LiDAR, vision and wireless are most commonly used for localization [43, 44, 62, 63]. Precise and robust localization is one of the critical requirements in such tasks as it forms the basis of robot navigation.

Many wireless-based localization techniques use trilateration and triangulation [41] for position estimation. Line-of-Sight (LoS) between the client and the Access Points (APs) is a requirement for these techniques. Due to LoS requirements, such techniques are not efficient for providing accurate indoor positioning as they lack LoS due to obstacles, moving components, and shelf partitions. Indoor wireless channels at high frequencies are more vulnerable to shadowing, atmospheric effects, and high attenuation due to materials [42]. So, such effects need to be considered while estimating the channel model, making radio channel estimation a non-trivial task. Further, high reflectivity and scattering from obstacles and the poor diffraction and penetration of mmWaves are the main factors preventing the reuse of known log-normal path loss models typically used for existing sub-6 GHz for high frequencies like 60GHz [64]. Additionally, the movement of workers, robots, and machinery inside the industrial building can affect the signal strength of wireless sensors significantly [65]. As a result, the wireless channel can become time-variant, which introduces more challenges in channel modeling [66,67]. Based on these challenges, fingerprinting-based approaches for wireless localization [34,68] are more suited for indoor environments.

Millimeter-wave channel in an indoor environment suffers from multi-path propagation [69]. To model the wireless channel, detailed knowledge of the electromagnetic characteristics from all scatters is required. Such modeling of indoor wave propagation at 60 GHz is non-trivial and involves custom hardware design. For this, the mmWave equipment used in our approach are the consumer-grade routers capable of communicating at 60 GHz frequency. Such consumer hardware is designed to communicate and provide coarse-grained channel state information due to cost-effective antenna array design [70].

In this work, we propose to use mmWave based wireless technology for wireless fingerprinting using Machine Learning (ML). We further enhance the localization performance of the mmWave system by integrating the ML prediction with a Kalman Filter (KF), *KF-Loc*. The ML model estimates static 2D position of the robot using the wireless features from 60 GHz mmWave routers. The sensor features within the complex indoor warehouse environment are susceptible to loss of connection due to shadowing effects and obstruction due to objects and shelves. This can result in the loss of position information of the robot, especially when the robot is in motion. ML models are trained using static data and alone cannot learn the robot's dynamics during motion. Further, the ML output is susceptible to misprediction due to the agent's motion. KF is designed and integrated with the ML model to overcome these challenges. The KF learns the motion of the agent and combines it with the ML output to provide highly accurate and smooth run-time tracking of the agent.

3.2 Contribution

The contributions of the proposed work are summarized as follows:

- **Machine learning based wireless localization:** We design an indoor warehouse localization system using consumer-grade off-the-shelf 60 GHz wireless routers. For this, we use Signal-to-Noise-Ratio (SNR) as a feature from consumer-grade wireless Access Points (APs) to create a radio map of the warehouse. The features from the radio map are used to train the ML model to learn the location estimation of an agent.

- **Date imputation and synthetic data augmentation:** We introduce a method to deal with the missing feature information from the APs during the training and inference of the ML models, as consumer-grade APs can lose connectivity intermittently, which can cause severe performance degradation. This approach to imputation is made by modeling the SNR characteristics using the collected dataset and further augmenting it with synthetic data.

- **Kalman filter design for run-time tracking:** ML models are trained using static data and cannot learn the robot dynamics during motion. Due to the robot's motion, the ML output is susceptible to run-time mispredictions. KF is designed and integrated with the ML model to overcome these challenges. KF learns the motion behavior of the robot and combines it with the ML position output to provide highly accurate and smooth run-time tracking of the robot.

3.3 Proposed mmWave Localization Architecture

We present our mmWave-based localization system, *KF-Loc*, for an indoor warehouse environment. Our localization approach uses multi-level ML models to estimate the location of the agent in the warehouse environment accurately. Figure 3.1 depicts the warehouse layout with the two aisles used in designing and evaluating the localization system. Within the two aisles, we have 68 distinct locations where we collect the SNR features. In our framework, we design a two-level ML approach for the location estimation where the first ML model performs the aisle level classification and the second ML model takes the signal information to regress the agent's position within the classified aisle.

The output from both the ML models are combined to estimate the position in the 2-dimension (2D) space. In our work, the agent moves in a single dimension, going down the aisle. This is realistic in many practical warehouses for autonomous forklifts where only one forklift is allowed to move in a single aisle at any given time in a unidirectional motion [33]. This is preferred as many tasks, including dispatching and path planning within the warehouse, aim to minimize the traffic and reduce collision for enhanced safety.

Wireless features used for location estimation are Signal-to-Noise Ratio (SNR) values from APs as captured by the client. The client, for communication purposes, only selects the AP with the highest signal strength. We enable the client to capture all available SNR signals by writing in-house firmware modification scripts. To build the dataset, we collect SNR features within the two aisles of the warehouse. The ML model uses the dataset to train network parameters during training stage. The wireless features are susceptible to obstacles and suffer from multipath reflections which can cause loss of connectivity with the receiver, which adds to the feature unreliability [69]. To overcome the before-mentioned negative impact on performance,

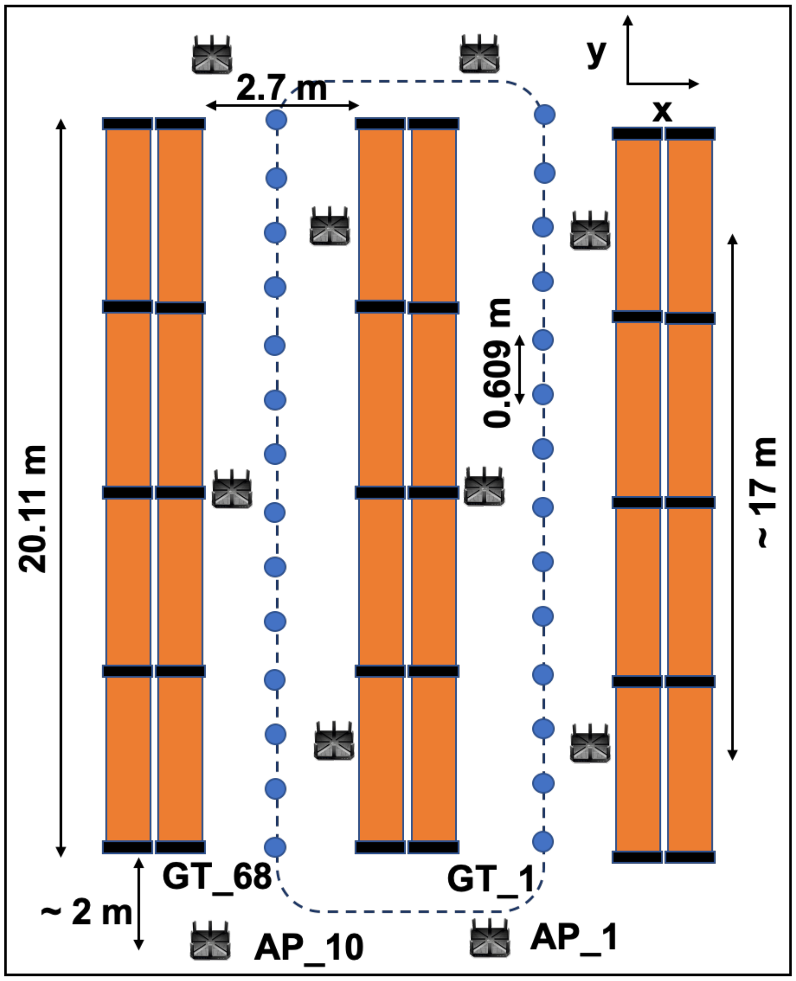


Figure 3.1: Warehouse layout with two aisles used as the testbed for system evaluation. The blue dots indicates the ground truth locations.

we introduce mean imputation and synthetic data generation technique as a pre-processing step. The details on data augmentation is presented in Section 3.3.3.

The localization system is divided into two different phases: an offline learning phase and an online inference phase. The framework is shown in Fig. 3.2. In the offline phase, first, the radio map of the warehouse is generated, then data imputation is performed to address the missing signal information in the radio map. The pre-processed radio map is then augmented with synthetic data to enhance the training and the robustness of the ML models. The training of ML models is performed in this phase. The trained ML models are deployed on the robotic agent during the

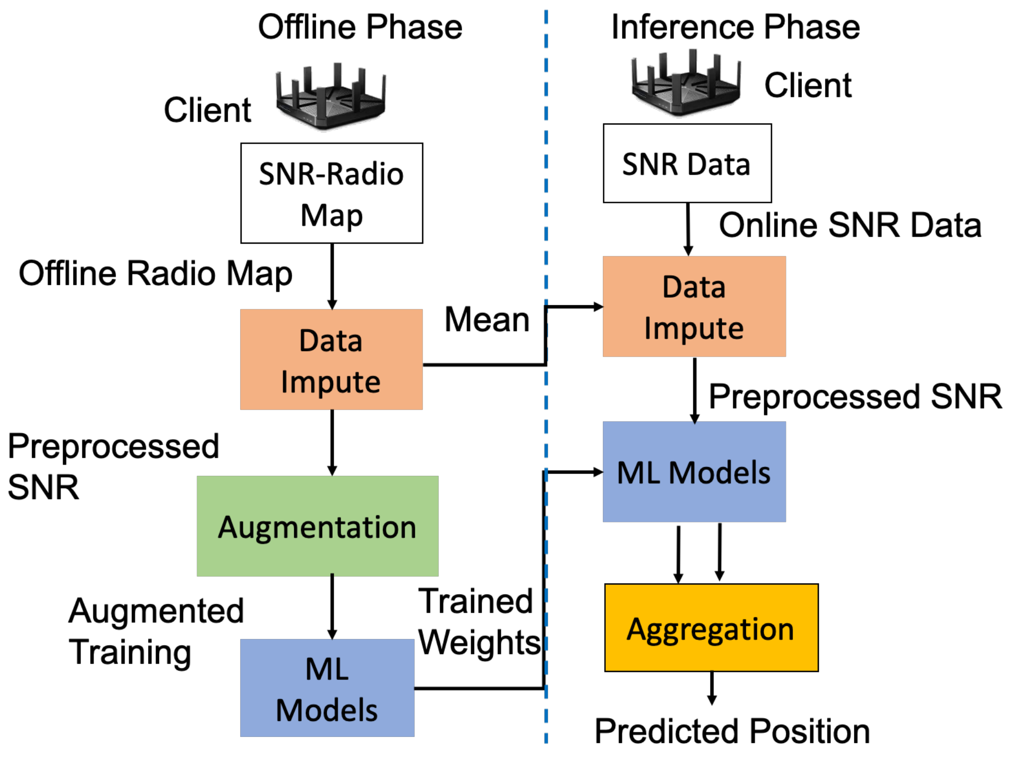


Figure 3.2: Implemented localization framework shows the two phases, the offline and the online phase. The offline phase is used during the training and the online phase for the run-time inference.

inference phase. The ML models are combined in a multi-level structure to provide the location estimate of the robot. In the following sections, we will discuss the design and working of each component in detail.

3.3.1 Access Points Deployment in the Warehouse

The APs used in our system are the 60 GHz TP-Link AD7200 wireless routers [71]. We selected these routers as they were the only available 60 GHz consumer-grade routers at the time of our experimentation. Figure 3.1 shows the placement of the APs on the warehouse's ceiling as the top view. The APs are mounted on the warehouse ceiling along the edges of the aisle in a zig-zag arrangement. This particular placement of APs is used to maximize wireless signal coverage within the aisle. Figure 3.3 shows the robotic agent with a 60 GHz router configured in the client mode. The routers

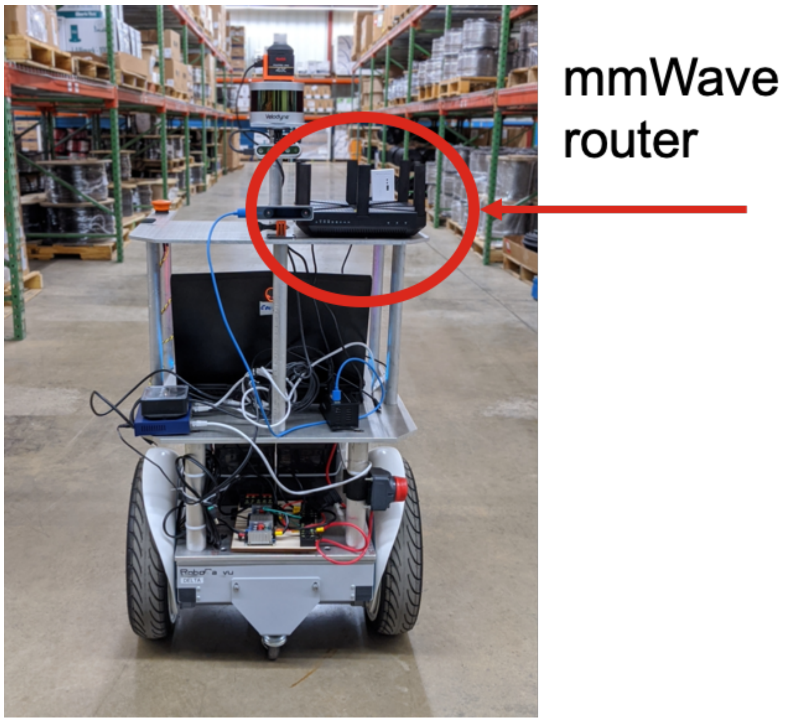


Figure 3.3: 60 GHz router configured as client on an robotic agent used in our work.

are configured in AP mode by default. Configuring the routers in client mode is done by flashing the router by the firmware provided by [70]. Also, as shown in Fig. 3.1, the total length of an aisle used in our experimental setup is 20.11 meters (66 feet), and we have placed five APs within the aisle.

3.3.2 Data Collection Process

To build the fingerprinting dataset, we collect the radio features within the two aisles in a functioning warehouse. Our training dataset is collected across different days and working hours in the warehouse. At each location, the client is programmed to scan for the SNR features from all available APs. This is done by interfacing the 60 GHz router in the client mode and then interfacing it with a computation unit that is present on the robotic agent. This additional step was required as, at the time of development, the laptop used for computation did not have a 60 GHz network card.

Next, the scan of the APs is done multiple times for each location within the aisles.

The rationale behind collecting data across multiple days and scans is to capture the random variations in the wireless signals across time and space. This routine makes the training data more feature-rich and significantly helps the ML models generalize with high accuracy to unseen data during the test time. During the offline phase, we pre-process the collected data to significantly improve the performance of the ML models and details of which are discussed next.

3.3.3 Wireless Data Augmentation

Many image-based data augmentation techniques are widely employed for ML applications such as image classification, object detection, and object recognition. These data augmentation techniques include image rotation, random cropping, and image color channel variations. The augmentation is performed for two primary reasons: first, to increase the size of the training dataset without physically collecting more data, which can save a significant amount of data collection time. Second, to increase the robustness of the ML models on unseen test data as it has been shown to drastically improve the learning capabilities of neural networks. But most of the data augmentation techniques used are for image-based datasets [72]. In our design, the features are the wireless SNR information and not image pixels, so the data augmentation techniques designed for images cannot be used on the wireless features.

In our approach, during the data collection, the client may be unable to capture the SNR information from all available APs. This is due to the interference and the shadowing effect that can occur with the high-frequency radio signals. Further, there can be situations when the SNR information from the sectors are missing, which we also observe while analyzing the collected dataset. The missing sector information and APs are inconsistent, and this can cause our model to not generalize for unseen test data. To overcome the missing SNR information, which can significantly impact the performance, we perform a mean imputation for each missing feature in the training

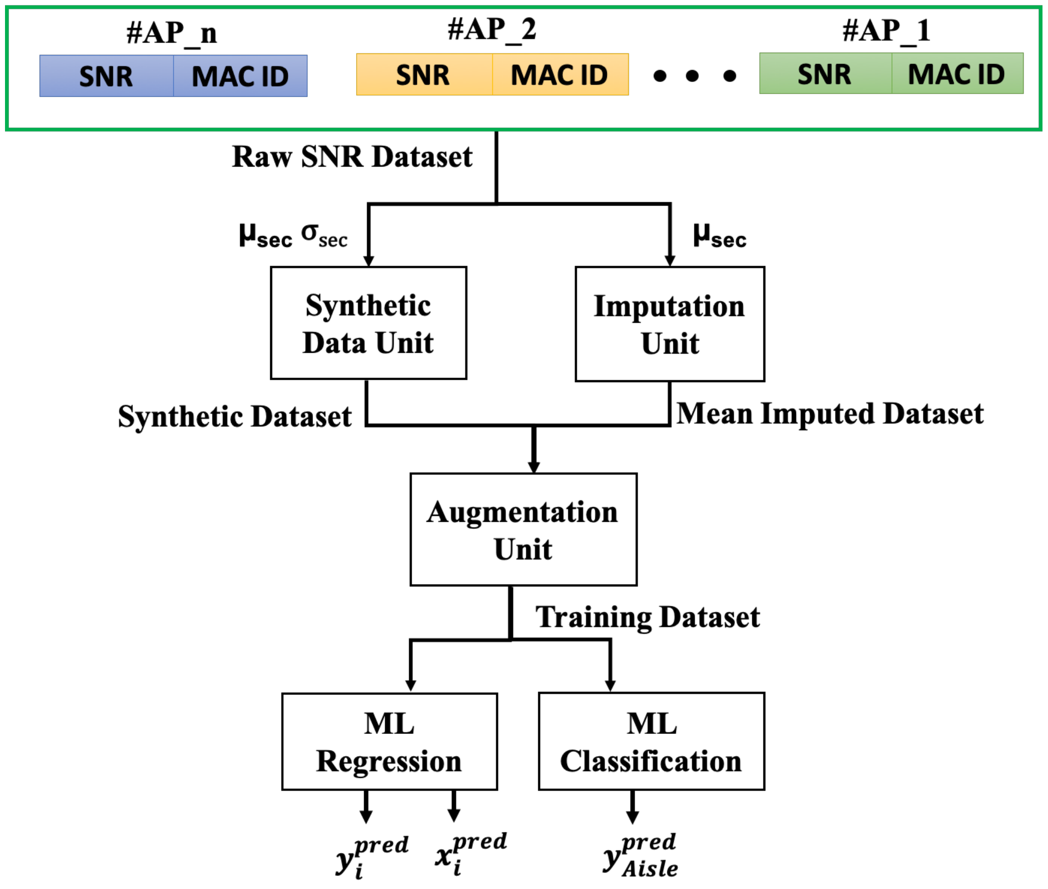


Figure 3.4: Detailed system architecture used during the offline training of the ML models.

dataset as a pre-processing step before we train the ML model.

For data augmentation, we present an approach to generate a synthetic SNR dataset by utilizing the SNR values from the APs. SNR estimation in communication systems with Additive White Gaussian Noise (AWGN) channel has been well studied in literature [73, 74]. In our mmWave hardware, the SNR measurements are extracted using the software patch provided by [70]. In our collected dataset, we observe the SNR density distribution as shown in Fig. 3.5, where we plot the density of sector SNR from an AP. To validate the Gaussian distribution of SNR measurements we perform the Anderson-Darling Test (ADT) [75–77]. The Anderson-Darling normality test is widely used to check if the input data samples follow the normal density distribution. In ADT [78], the hypothesis that the data belong to Gaussian distribution is rejected

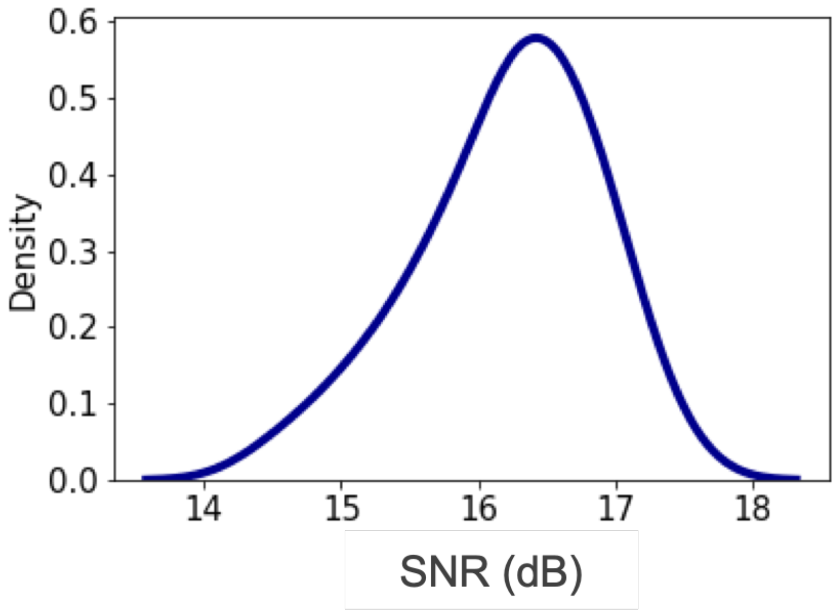
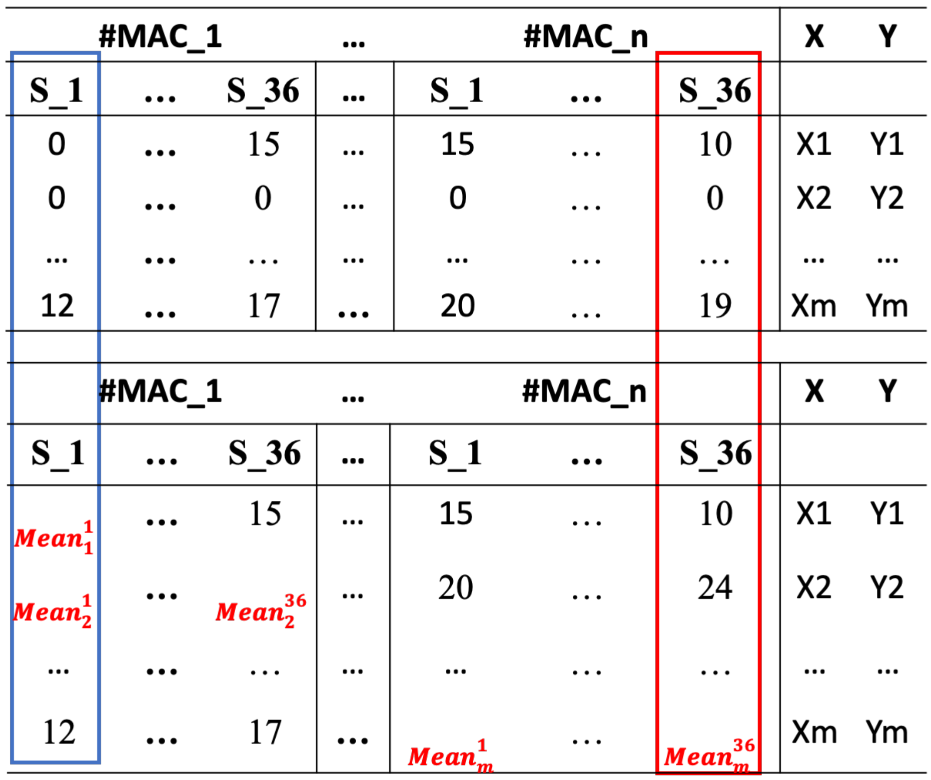


Figure 3.5: SNR feature density distribution using the collected dataset inside the warehouse

if the computed test statistics is greater than the critical value at 5% significance level. Based on the experiment, for the SNR density as shown in Fig. 3.5, the test statistics is found to be 0.47, which is below the critical value of 0.70 at 5% significance level. As the test statistics value is lower, we conclude that it fails to reject the null hypothesis. Using this we assume the Gaussian distribution for the SNR features. A similar approach is also used for synthetic augmentation work using angle-of arrival data for mmWave [77]. Where wireless features used in the localization system are augmented with synthetic data after performing the ADT normality test.

We model the mean of SNR value from an AP j for sector i, $\mu_{i,k}^{j}$, and standard deviation $\sigma_{i,k}^{j}$ from the collected dataset at location k. In this way we artificially create a new synthetic dataset which help reduce the data collection time that can be fairly significant for large warehouses. Next, after performing the imputation and synthetic data generation, we augment the on-site imputed data with the synthetic data. This enables us to improve the learning capability of the ML models and



#MAC_1			...	#MAC_n			X	Y
S_1	...	S_36	...	S_1	...	S_36		
0	...	15	...	15	...	10	X1	Y1
0	...	0	...	0	...	0	X2	Y2
...	...	...	...	...	...	...	...	...
12	...	17	...	20	...	19	Xm	Ym

#MAC_1			...	#MAC_n			X	Y
S_1	...	S_36	...	S_1	...	S_36		
$Mean_1^1$	...	15	...	15	...	10	X1	Y1
$Mean_2^1$	...	$Mean_2^{36}$	...	20	...	24	X2	Y2
...	...	...	...	...	...	...	...	...
12	...	17	...	$Mean_m^1$	...	$Mean_m^{36}$	Xm	Ym

Figure 3.6: Output data from Imputation Unit that is used for mean substitution for the missing SNR features in the collected dataset

the robustness of the trained models when performing the test-time inference. The process is also illustrated in Figure 3.4, where the Imputation Unit (IU) extracts the mean and performs the imputation for the missing sector information on the collected dataset. Figure 3.6 shows the output data from the IU. Here, the missing SNR values are imputed based on the mean value of individual features at each location. The Synthetic Data Unit (SDU) extracts the features' mean and standard deviation from the raw dataset and creates another set of synthetic datasets. The Aggregation Unit (AU) combines the raw and synthetic datasets to create the final training dataset, which is used for training the ML localization models. Next, we will discuss the design of the different ML models used in our localization system.

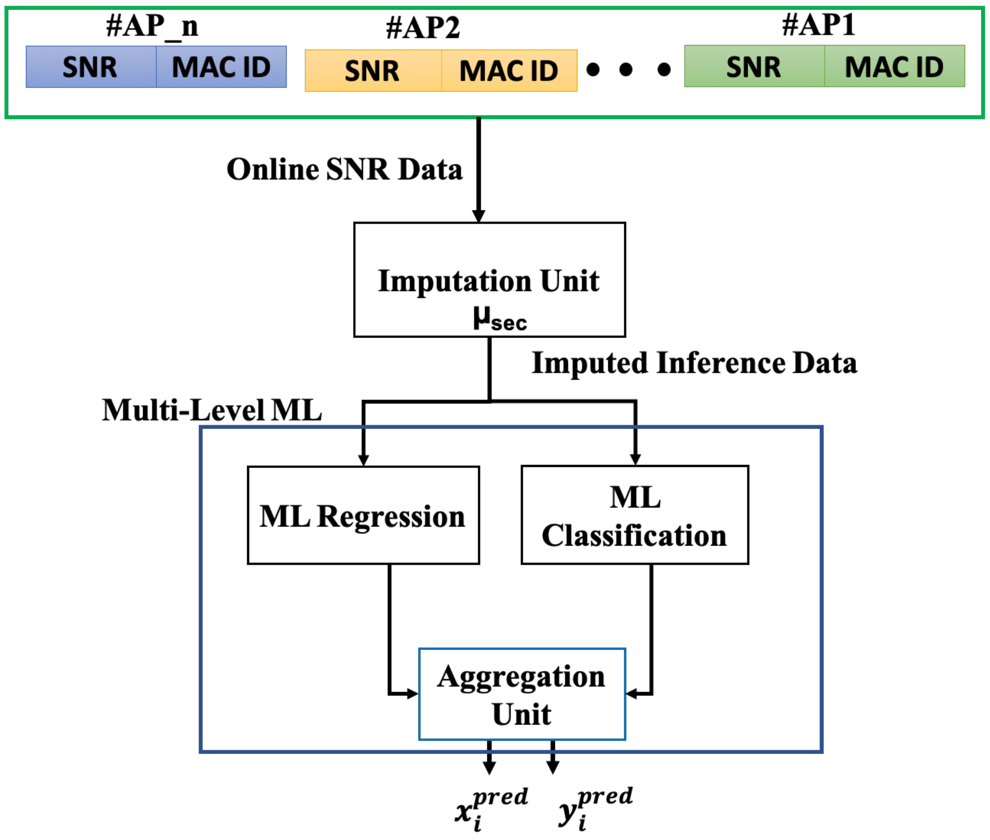


Figure 3.7: Test time inference framework for localization that is used during the online phase.

3.4 Multi-level Machine Learning for Localization

The task of the localization algorithm is to learn the complex wireless features from the 60 GHz APs as a function of distance with the mobile client with high accuracy and robustness. We implement a multi-level ML-based localization system to achieve high localization accuracy within the aisles and a robust learning algorithm. In the first level, we perform the aisle level classification within the warehouse, which makes our implemented system scalable to many aisles. Within the aisles, we obtain coarse-level information regarding the agent's position, and we can easily learn the relationship of the features as a function of aisles with high accuracy. The next level of ML learning is the regression-based approach, where the ML model learns the position in a 2D space. The regression model outputs a continuous valued position coordinates of the agent within the aisle. The models are trained independently during the offline training phase.

During the inference phase, the predictions from ML classification and the ML regression are combined together as a post-processing stage, then based on the aisle level information from the ML classifier the X-position of the agent is adjusted. The Y-position information from the ML regression remains unchanged. The inference setup is illustrated in Fig. 3.7 where at the run-time the instantaneous SNR signal values from the APs are observed at the client at every time step the agent is in motion, then the SNR data is pre-processed by the IU, this is done at the inference time as the models are trained on the augmented data, so ML models expects the consistent and same data structure during the inference phase. The IU is very light in terms of the computational complexity as it only receives the instantaneous SNR values and impute them with the saved mean information.

For both ML models, the input dimension is a vector of 360 features denoted by X_{fet}, where $X_{fet} = \{ x_0, ... , x_{359}\}$. Where, x_n represents the SNR value from APs. The output for the classification is an integer value for all the locations the data was collected during the data collection routine. In our setup we have 68 distinct locations and each row in the dataset will have one of the location as the output. In case of the regression, the output is a 2D vector representing the X and Y-position of the agent. Next, we discuss the design of the ML classification and the ML regression models used for localization.

3.4.1 ML Classification

The first level in our localization system is the aisle-level location estimation of the agent within the warehouse. In this level, the ML classification model is trained on the collected radio dataset as described in Section 3.3.2. We define different aisles as distinct class labels to train the ML classification model. With this, we aim to achieve the aisle-level position estimation of the agent, providing us with the agent's coarse-level location. The ML regression model then uses the aisle level position information to predict the agent's location in 2-D space more accurately. The output depends on the number of aisles in the warehouse. Next, we will discuss the ML regression approach toward multi-level learning.

3.4.2 ML Regression

After the design of the ML classification for estimating the aisle level positioning, we design the ML regression model for providing the continuous position estimates within the aisle. The regression approach uses the outputs as the 2D coordinates of the agent within the warehouse. Here the outputs are the continuous values with two output heads as shown in Fig. 3.4. Similar to the ML classification model, we train the model with the same input data after augmentation. The ML regression performs the fine-tuning of the agent position within an aisle. In the final localization system, we use the aisle level information from the ML classification model to locate the aisle where the agent is in and then use the Y-position information from the ML regression model to narrow down the agent's position within the aisle.

3.5 Kalman Filter Design

Kalman filter estimates the state of a robot based on the prediction-update cycle. KF is a linear recursive estimator that minimizes the mean square error of the estimated parameters. In our design, we use KF to track the agent's position in a 2D space within a warehouse. The input to the KF is the noisy position output from the ML localization system, i.e., (x,y) coordinates of the robot. The KF filters the noisy coordinates to generate a more accurate 2D tracking. First, the KF is initialized by setting up the various parameters. This includes the setup of the 60 GHz sensor's noise covariance matrix (R) and measurement matrix (H), along with the initialization of the robot's state vector (x), state transition matrix (F), and covariance matrix (P). We assume the process covariance (Q) to be zero as we perform a constant velocity-based linear tracking of the robot.

In the first step, the KF performs the prediction of the robot's state and error covariance for the next time step, and this is represented using (1), (2). Where B and u represent the control input matrix and control vector due to the external and internal forces. In the next step, KF performs the update process where it updates the previously predicted state and covariance estimate based on the received measurements. This is shown mathematically by (6), and (7). During the prediction stage, the position uncertainty of the robot increases, as no information is gained during this stage. While during the update step, we gain information through the sensor measurements and we become more certain regarding the robot position.

$$\hat{x_k} = Fx_{k-1} + Bu_{k-1} \tag{3.1}$$

$$\hat{P_k} = FP_{k-1}F^T + Q \tag{3.2}$$

$$y = z_k - H\hat{x_k} \tag{3.3}$$

$$S = H\hat{P_k}H^T + R \tag{3.4}$$

$$K = \hat{P_k}H^TS^{-1} \tag{3.5}$$

$$x_k = \hat{x_k} + Ky \tag{3.6}$$

$$P_k = (1 - KH)\hat{P_k} \tag{3.7}$$

KF considers the position probability of the robot to be a Gaussian probability density function (PDF) which can be characterized by the mean and the standard deviation. The process starts by initializing the belief of the robot at the start time. For this, we take in our initial estimation output from the ML model. Next, using a motion model for the system, which is a constant velocity model in our design, we estimate the location of the robot in the next time step. During the update cycle of KF, after the time step has elapsed, we obtain the measurement reading from the ML model and update our previously predicted belief of the robot state. It is to note that the belief of the robot position, i.e., the state and the measurements, are modeled as Gaussian PDF.

Multiple estimates from various localization sources can be fused together to further improve the localization performance with the Kalman Filter. Fusion of different localization systems is also referred to as sensor fusion [79, 80]. In this work, we only utilize mmWave router as a localization sensor without sensor fusion.

3.6 Experimental Analysis

In this section, we will discuss the performance of our implemented ML-based localization system. For our testbed, we have used a working warehouse [81]. Inside the warehouse, we have selected two different aisles where we deploy 10 APs on the ceiling, as shown in Fig. 3.1, in a zig-zag placement. For APs and clients, we have used 60 GHz TP-Link AD7200 routers. During the data collection, we collected the training data within the two aisles by moving the agent 0.609m (2 feet) in the y-dimension of the aisle and keeping the x-dimension fixed. We have 68 distinct positions where we collect the data for the two aisles. We perform the wireless scan at each location 10 times at the client. For both the aisles, we collect two different training datasets. The datasets are collected within multiple days to capture the temporal variation of the wireless communication links.

The training dataset used by ML localization models consists of 1340 samples corresponding to 68 GT locations. For testing, we use a separate dataset consisting of 680 samples. For Kalman Filter evaluation, we consider one data point at each of the GT location from the test dataset within the two aisles. Where, assuming constant velocity model for the agent each of the data point is considered one time

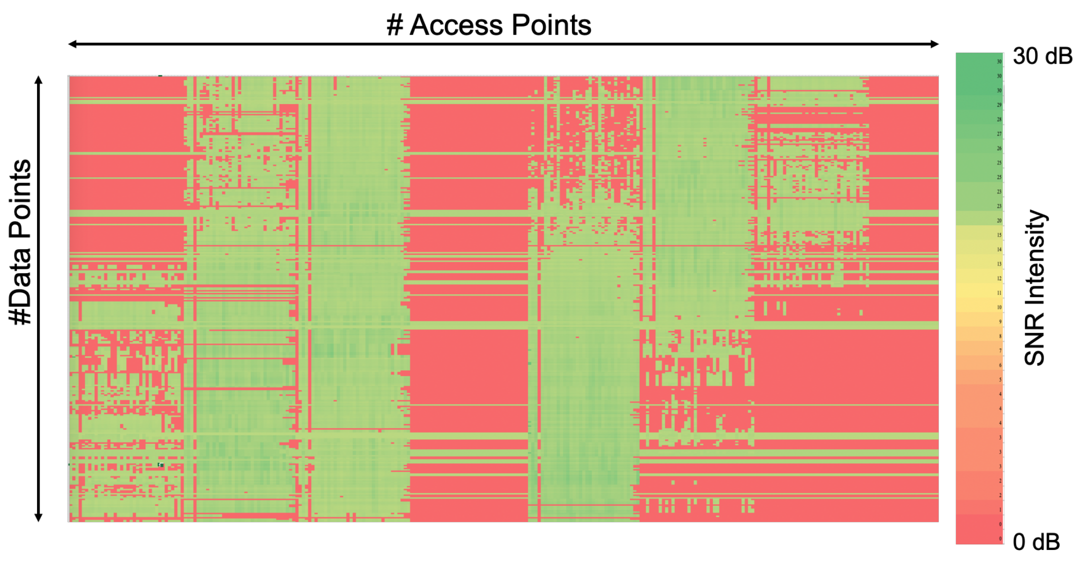


Figure 3.8: Color coded SNR radio map of a single warehouse aisle recorded at different locations

step unit apart. Further, in our evaluation, we have considered different kinds of ML models for both classification and regression. For classification, we have selected K-Nearest Neighbor (KNN), Support Vector Machine (SVM) and Multi-Layer Perceptron (MLP). For ML regression, we have used Support Vector regression (SVR), Linear regression (LR) and MLP.

3.6.1 Performance With Imputation and Augmentation

In our KF-Loc system, we perform imputation and synthetic data augmentation during the offline training phase as discussed in Section 3.3.3. Here, we compare the performance between the different ML models trained with and without synthetic data augmentation. Figure 3.8 illustrates the SNR radio map recorded at the warehouse during the data collection process. The figure shows the raw data collected without mean imputation. The color coding indicates the varying SNR intensity at different locations as seen on the client from all APs. The red color indicated that no data was recorded at the client and we term them as missing SNR values.

We see significant performance improvement after performing data augmentation

Table 3.1: ML regression model performance with data augmentation

ML Model	**Configuration**	**Augmentation**	
		With	**Without**
LR	Linear	(0.51m, 2.60m)	(0.58m, 2.84m)
SVR	Polynomial Kernel	(0.71m, 2.17m)	(0.48m, 2.33m)
MLP	200, 200, 200	(0.19m, 0.92m)	(0.27m, 1.54m)

and imputation processing on training dataset. For ML regression models, Table 3.1 shows the performance comparison between different regression models trained with and without data augmentation. MLP regression model achieves significant performance improvement, with RMSE of 0.19m and 0.92m in X-and Y position respectively. This shows 29.3% and 40.25% improvement when compared with the RMSE performance without data augmentation based training. For the other two regression models, LR and SVR, performance improvement of 8.4% and 6.8% is achieved in Y-position respectively. This improvement in performance is seen due to robust training of the ML models with the synthetic data augmentation and mean imputation.

Table 3.2, represents performance comparison between different ML classification models trained with and without data augmentation. Performance improvement is seen for the MLP and SVM classification models. For our optimized SVM classification model performance improved from 95.70% to 99.55% which shows the significance of the data augmentation based training process. Overall, the data augmentation approach that is introduced in our localization system benefits both regression and classification based ML models.

3.6.2 Multi-Level Localization System

Here, we analyze the ML classification and regression models used in the presented localization system. For both classification and regression models, we evaluate and compare performance of different types of ML models. Next, we integrate the KF with the multi ML localization model, where the static estimates are used to perform dynamic run-time tracking.

3.6.2.1 ML classification analysis

We design an ML classification model to predict the aisle level position estimate. For this, we train and test different classification-based ML models. We evaluate the performance of three different ML classification models, KNN , SVM, and MLP. These three are selected as they represent varying complexity and network architectures. KNN estimates the position by considering the K closest input features. SVM works by selecting a hyperplane such that the margin around the plane is maximized to separate the data points for different classes. Kernels are used to map input features into higher dimensional space. The MLP model consists of fully connected layers. MLP is more complex than SVMs as they can learn a large number of parameters for input with very high dimensions.

Table 3.3 represent the performance evaluation of the three ML classification models on the test dataset. We see all three models achieve high classification accuracy. The high classification accuracy is achieved as the position information at the aisle level is the coarse-level localization problem, and the features learned by the ML models are able to provide high confidence mapping between the aisle position and the wireless features.

Table 3.2: ML classification model performance with data augmentation

ML Model	**Augmentation**	
	With	**Without**
KNN	98.37%	98.37%
SVM	99.55%	95.70%
MLP	99.25%	97.47%

Table 3.3: Performance comparison with different ML classification models

Model	Configuration	Accuracy	Precision	Recall	F1 score
SVM	**RBF, C=20**	**99.55%**	**1.0**	**1.0**	**1.0**
KNN	K=3	98.37%	0.98	0.98	0.98
MLP	50	99.25%	0.99	0.99	0.99
MLP	100,100	98.10%	0.98	0.98	0.98

KNN, a simple complexity ML model, can achieve 98.37% classification accuracy, and the MLP achieves an accuracy of 99.27%. Compared to MLP and KNN, SVM achieves the highest classification accuracy of 99.55% and achieves better precision, recall, and F1-score than the other two models. Due to the high performance achieved by the SVM for providing aisle-level position estimates, we use SVM as our ML classification model in localization architecture. Next, we discuss the design of the ML regression model that estimates the 2D position of the agent within the aisle.

3.6.2.2 ML regression analysis

Here, we present the analysis of the ML regression model used in our localization architecture. The regression-based learning estimates the real-valued continuous output in 2D space, compared to discrete class labels in the classification. Here, we train and evaluate different regression models to select the best performing ML regression model. For this, we trained and evaluated three types of ML regression models.

Table 3.4 illustrates the performance of the three ML regression models. The LR model has poor performance with a Y-position Root Mean Square Error (RMSE) of 2.6m. It shows that using LR, a simple and less complex learning model, the complex behavior of the 60 GHz wireless features cannot be learned with high confidence and requires more complex learning models. In comparison, SVR and MLP both perform with high accuracy. An MLP-based regression model with three hidden layers of 200 neurons each achieves the highest accuracy of 0.92m in Y-position and 0.19m in X-

Table 3.4: Performance comparison with different ML regression models

Model	Configuration	RMSE-X	RMSE-Y
LR	Linear Model	0.51m	2.60m
SVR	Polynomial kernel	0.71m	2.17m
MLP	**200, 200, 200**	**0.19m**	**0.92m**

position. The MLP also achieves the lowest median error of 0.26m compared to the SVR and LR regression models. Figure 3.9 shows the localization error performance using the Cumulative Distribution Function (CDF) plot, where the three different regression models are compared. From the plot we obtain lowest median error of 0.26m for MLP.

Table 3.5, shows the optimization process used to obtain the best performing ML regression model. Here, we have changed the MLP configuration by varying the number of hidden layers, the number of neurons in each hidden layer, and the learning rate. The parameters and hyperparameters tuning is required during the optimization of ML models. The performance of the optimized model achieves test RMSE of 0.19m and 0.925m in X-and Y-position, respectively. The model also achieves a Mean Absolute Error (MAE) of 0.47m, hence achieving the centimeter-level localization accuracy within the warehouse aisles.

We also consider the fourth metric that shows the performance for achieving less than meter level accuracy. The optimized model outperforms all other configurations by achieving 97.03% estimation with error less than 1m. We have considered a total of 674 test cases to evaluate this percentage. Based on these performance metrics, we have selected the three hidden layer MLP model with 200 neurons each as our ML regression model for the localization system. From Table 3.5, we also observe that when we go for deeper models with more than three hidden layers, the model's accuracy decreases compared to our optimized model. This is expected as the number of learning parameters increases significantly when we go towards deeper ML models.

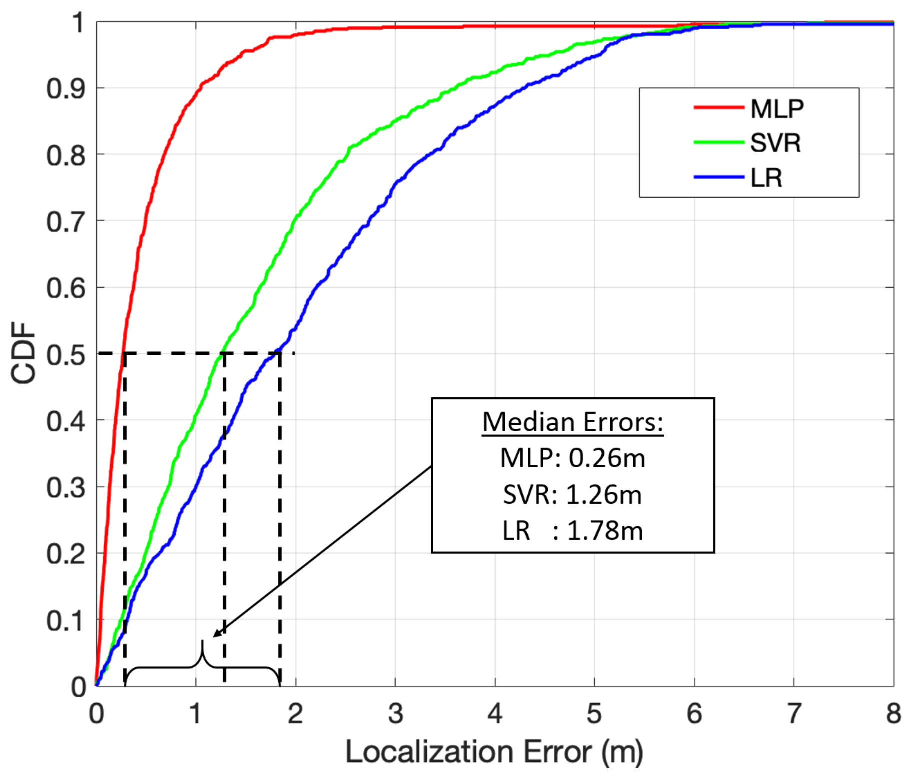


Figure 3.9: Localization performance for different ML regression models using cumulative density function. The optimized MLP achieves median error of 0.26m

The model shows overfitting for the deeper models and underfitting for the shallow models. Figure 3.10 shows the position output of ML regression model in the two test aisles. It can be seen that the estimated locations are very close to the actual location. The estimation in X-position shows scattering between the two aisles. This is expected as here we are predicting continuous valued output with very low resolution along the X dimension. Further, in the final system we use the ML classification output to represent the X-position of the agent and use the Y-position from the output of ML regression model.

Table 3.5: MLP regression optimization

Configuration	**Activation**	**RMSE-X** (m)	**RMSE-Y** (m)	**MAE-Y** (m)	**Error-Y** $<$ **1m**
50	ReLu	0.43	1.77	1.29	59.94%
200	ReLu	0.36	1.57	1.13	86.20%
100, 100	ReLu	0.21	1.03	0.60	93.32%
200, 200	ReLu	0.23	1.27	0.69	90.36%
100, 100, 100	ReLu	0.22	1.02	0.56	94.36%
200, 200, 200	**ReLu**	**0.19**	**0.92**	**0.47**	**97.03%**
100, 100, 100, 100	ReLu	0.19	1.05	0.50	95.25%
200, 200, 200, 200	ReLu	0.20	1.21	0.56	94.21%

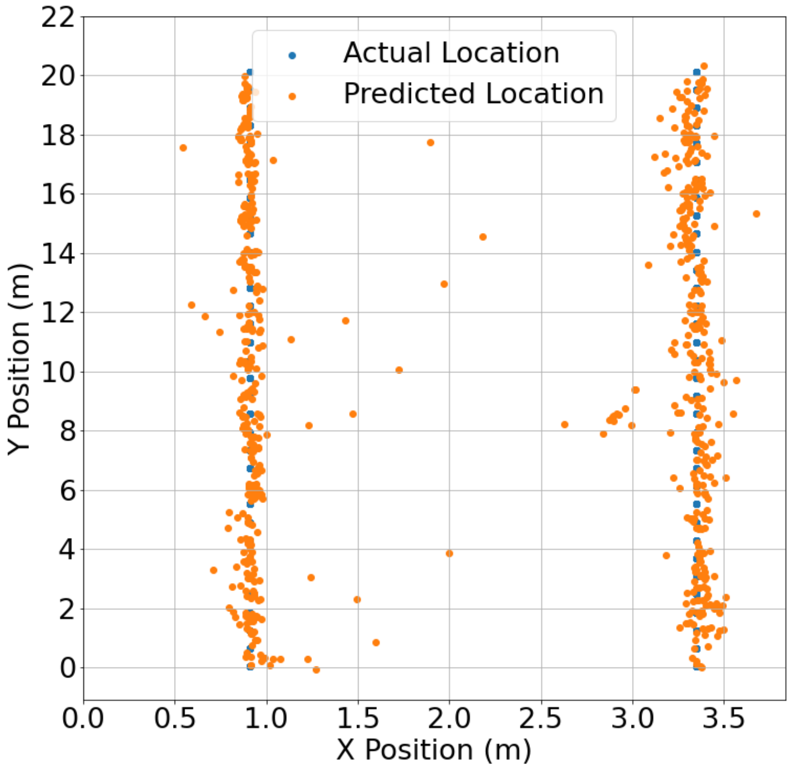


Figure 3.10: Scatter plot between actual and predicted position by optimized MLP regression in two aisle

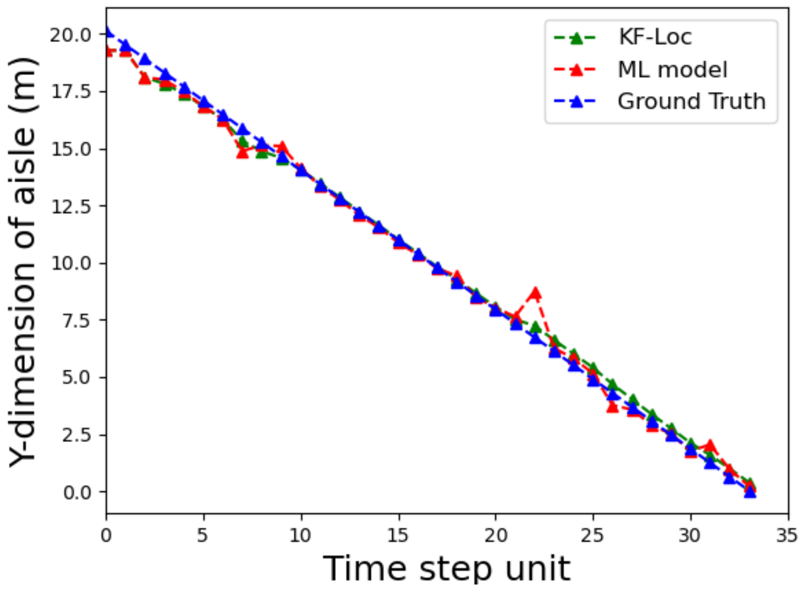


Figure 3.11: Comparison of y-position between KF-Loc and ML model in aisle-1

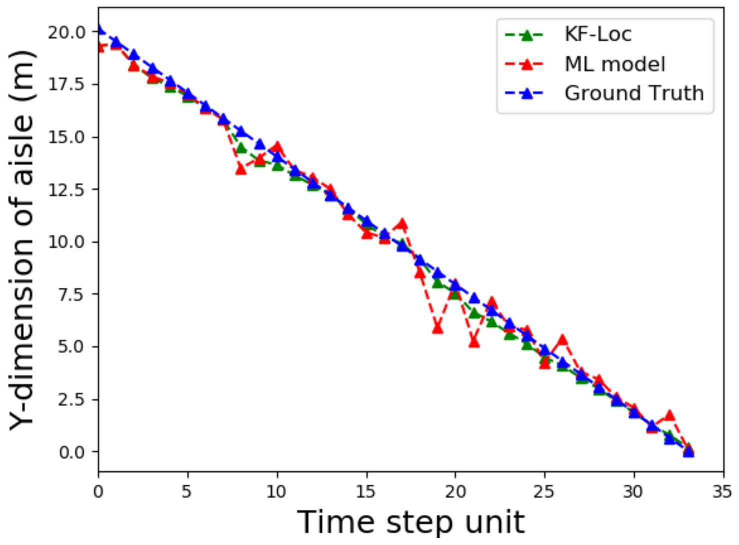


Figure 3.12: Comparison of y-position between KF-Loc and ML model in aisle-2

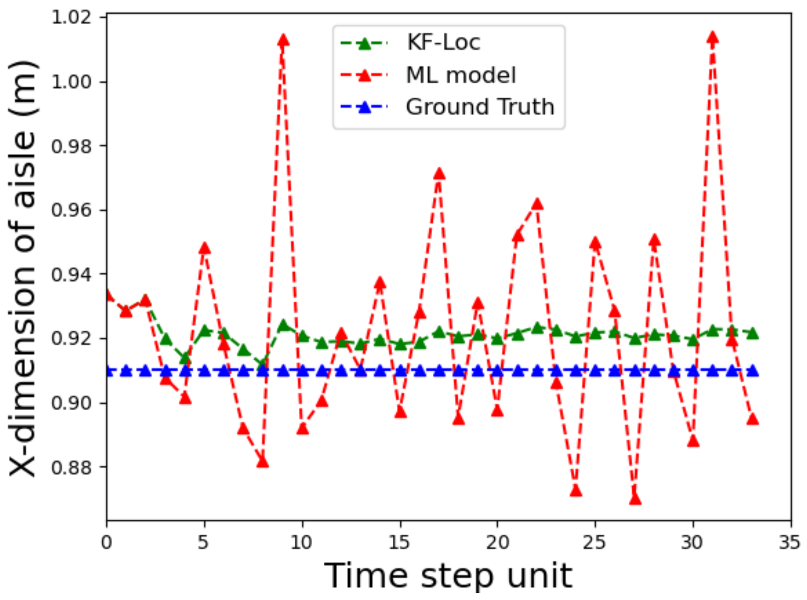


Figure 3.13: Comparison of x-position between KF-Loc and ML model in aisle-1

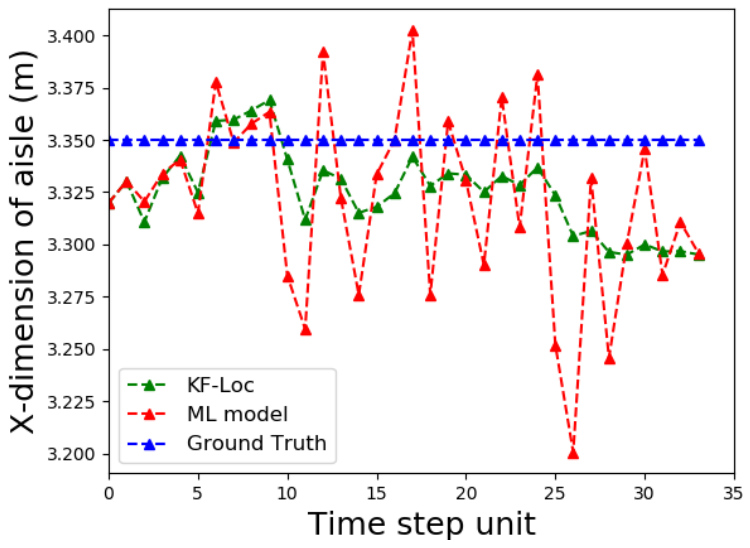


Figure 3.14: Comparison of x-position between KF-Loc and ML model in aisle-2

3.6.3 Kalman Filter Integrated Tracking

Here, we integrate the KF system with the ML localization models. We define the robot state as a three-dimension vector defining the position in the 2D Cartesian co-ordinate frame and its velocity in the y-dimension. The covariance matrix, P, is a 3x3 matrix initialized with very high uncertainty for velocity. The measurement matrix, H, is a 2x3 matrix, initialized with 1 for the position, and the measurement covariance, R, is a 2x2 matrix initialized with values 0.01. The ML output is integrated with the KF to provide the 2D tracking.

Figures 3.11 and 3.12 show the performance comparison between the KF-Loc, in green, and ML model, in red, tracking in y-position for aisle-1 and aisle-2 respectively. The GT position is shown in blue for both figures. It can be seen that the ML model produces fluctuations in the position estimates as the robot moves along the aisles. The fluctuations are due to the mispredictions by the ML model. These mispredictions are caused by the signal fluctuations at the client due to the robot's motion, resulting from shadowing effects, and obstructions between the client and the APs [69].

Similarly, Figs. 3.13 and 3.14 shows the position estimates of the robot along the X-position within the aisle. Similar fluctuations are seen due to the dynamic motion of the robot, causing mispredictions in the position estimation by the ML model. For location estimation in both dimensions, the KF-Loc system provides much smoother localization and tracking performance. This is seen as the KF can filter and smooth out the raw position estimation from the ML model and capture the robot's dynamic motion by estimating its velocity. This is where the KF provides powerful integration with the ML model by estimating the robot's dynamic state variable, in our case, the velocity, without explicitly being provided with velocity output from the robot. Thereby providing a more robust and reliable run-time motion tracking.

The performance of KF-Loc is compared with the ML model-based localization system by measuring the robot's Root Mean Square Error (RMSE) in both the aisles.

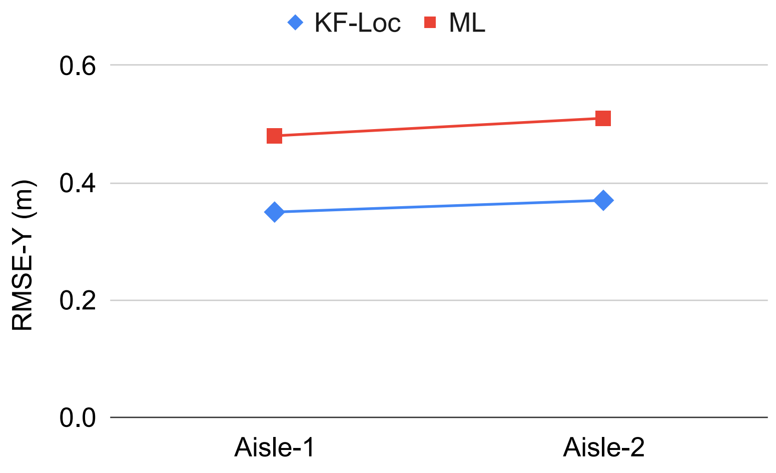


Figure 3.15: Error comparison between KF-Loc and ML model in both the aisles

Figure 3.15 plots the robot's RMSE in Y-position for aisle-1 and aisle-2. With KF integration, a 28.5% and 54.3% reduction in RMSE is achieved compared to ML-based localization systems in aisle-1 and aisle-2, respectively. Also, for both the aisles, KF-Loc achieves centimeter-level localization accuracy with RMSE of 0.35m and 0.37m, respectively.

3.7 Comparison with Different Localization Approaches

For detailed evaluation, we compare the performance of our proposed KF-Loc with different wireless-based localization approaches. Table 3.6 compares our work with many different state-of-the-art approaches to wireless localization systems using millimeter wave and sub 5GHz features. Work presented in [82–84] uses lower 2.4GHz frequency and provides localization using K-nearest neighbor [82], artificial neural network (ANN) [83], and fingerprinting matching [84] based approaches. The performance achieved with these approaches shows sub-meter level accuracy. Further, work presented in [24, 70, 85, 86] utilized mmWave frequencies varying from 28GHz to 60GHz. They achieve meter-level positioning accuracy, and all the approaches are static estimations and are susceptible to high errors during run-time tracking in the

Table 3.6: Performance comparison with different wireless based localization techniques

Work	Wireless	Frequency	Environment	Methodology	Performance
Bahl [82]	RF-based	2.4 GHz	Indoor	KNN	2m-3m
Laoudias [83]	WiFi	2.4 GHz	Indoor	ANN	Mean error of 3.4m
Yang [84]	WiFi	2.4 GHz	Indoor	WiFi Fingerprinting	Mean error of 5.88m
Kanhere [85]	mmWave	28 GHz	Indoor	Fusion of AoA and received power	Mean error of 1.86m
Kanhere [85]	mmWave	28 GHz	Outdoor	Fusion of AoA and received power	Mean error of 34m
Bielsa [70]	mmWave	60 GHz	Indoor	Particle filter	Median error of 1.1m to 1.4m
Wei [86]	mmWave	60 GHz	Outdoor	DoA based WKNN fingerprint	Mean error 1.32m
Walaa [87]	mmWave	28 GHz	Outdoor	Different ML models	MAE of 3m-33m
Li [88]	WiFi	5 GHz	Indoor	Particle Swarm Optimization	Median error 1.5m
Vashist [24]	mmWave	60 GHz	Indoor	MLP fingerprint	RMSE 0.84m
Our work	**mmWave**	**60 GHz**	**Indoor**	**KF and ML integrated**	**RMSE of 0.35m and 0.37m** **MAE of 0.47m** **Median error of 0.26m**

event of mispredictions. These mispredictions occur due to the multipath effects in higher frequencies.

Authors in [87] experiment with a simulation-based environment using a 28 GHz based fingerprinting approach and evaluated the performance using 13 different ML models. They achieve meter-level accuracy for different ML models in an outdoor simulation environment. Further, the results are simulation based and not evaluated on a real testbed. Li [88] uses 5GHz WiFi based location estimation using Particle Swarm Optimization (PSO) approach. They report a median positioning error of 1.5m. Work done by authors in [88] utilizes an office space as their experiment testbed. The work lacked discussions regarding how their system will perform in a more complex dynamic environment like a warehouse. Our approach utilizes dynamic learning by integrating KF with the ML position estimation. As a result, we achieve significant improvement in performance with 0.35m and 0.37m RMSE compared to all mentioned approaches using sub-5GHz or millimeter-wave frequencies as shown in Fig. 3.15. The figure further illustrates the error accumulation that can occur in location estimation models that only produce static estimates and how our KF-Loc can significantly reduce the negative impact of miss-predictions caused by the ML models.

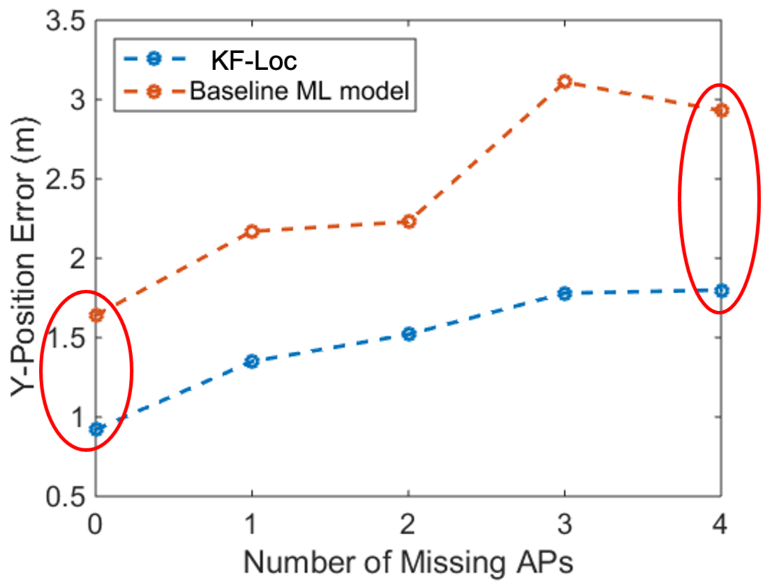


Figure 3.16: Robustness performance of KF-Loc system with baseline ML model for regression

3.8 Robustness with Data Augmentation

Here we compare the performance of the KF-Loc system with the ML localization model trained without data augmentation and KF integration. The ML model is the same MLP that we use in the KF-Loc. To evaluate the robustness performance, we randomly switch off data corresponding to APs, introducing missing or corrupt wireless signals in the test dataset, capturing the scenario where APs can lose connection with the client. Figure 3.16 illustrates the performance where the x-axis represents the number of missing APs, and the y-axis shows the corresponding RMSE in the y-dimension.

We see the KF-Loc system maintains a lower positioning error compared to the ML-based system, with RMSE of 1.98m and 2.93m, respectively. The significant decrease in performance for ML-based systems is due to the low robustness of the trained model. In comparison, KF-Loc maintains high tolerance to signal fluctuations due to the augmentation-based training. Data augmentation makes the localization

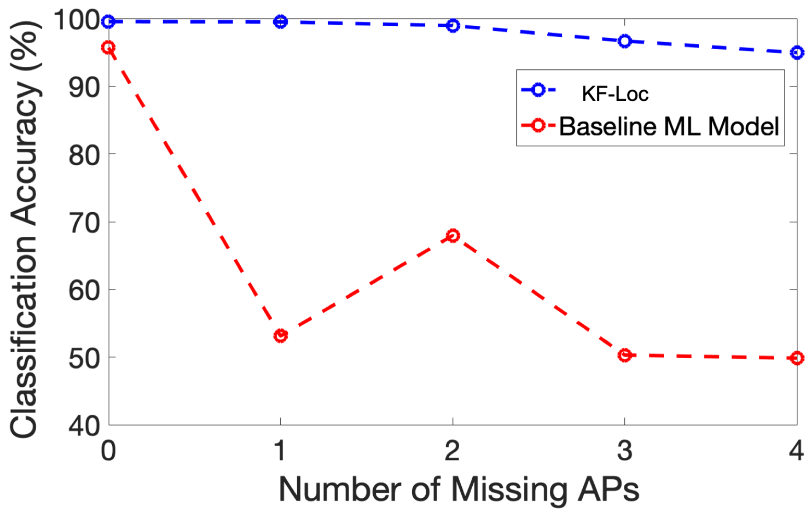


Figure 3.17: Robustness performance of KF-Loc system with baseline ML model for classification

system highly robust against random fluctuations and loss of connectivity with the wireless signals. Such scenarios are common in real-world deployment using ML-based localization systems, and the ability to generalize with high robustness is a critical requirement. Figure 3.17 shows the robustness with the ML classification model used in the KF-Loc. We again observe the high robustness compared to the ML classification model trained without the data augmentation. The KF-Loc maintains the high classification accuracy of 97% with four AP dropouts, while the accuracy of the ML model decreases to 50%. Such robustness performance analysis is missing in related state-of-the-art localization models presented in Table 3.6.

3.9 Conclusion

We present the design and implementation of a robust localization system, *KF-Loc*, for indoor warehouses using 60 GHz routers. We introduce the use of the consumer-grade 60 GHz wireless routers for providing high accuracy localization performance using off-the-shelf mmWave routers. In our system, complex mmWave features are learned by a multi-level ML models, providing static position estimations of the robot. The aisle level position is provided by a ML classification model and the position of the agent within the aisle is provided by the ML regression model. Kalman filter is designed to improve the ML estimation error during the robot motion. KF improves the motion tracking of the robot by removing the fluctuations due to mispredictions in ML output. ML models are susceptible to fluctuations in input features causing severe performance degradation. To address robustness, we present a data imputation and augmentation for wireless features. Our results show for the worst case AP dropout, KF-Loc achieves 1.4X less degradation in localization performance. To test the practicality of our system, we deploy and test our system within two aisles in a functional warehouse. KF-Loc achieves centimeter-level accuracy in our test setup of two aisles with RMSE of 0.35m and 0.37m, respectively. Further, when compared with the static ML localization system, our proposed system shows significant performance improvement by achieving 28.5% and 54.3% improvement in RMSE error for the two test aisles.

3.10 Chapter Summary

Increasing demand of industrial automation has led towards the next generation of industrial revolution, Industry 4.0. For warehouse management, intelligent automation robots are required to perform various tasks like monitoring, scheduling and dispatching. One of the most fundamental and critical challenge for such autonomous robots is the task of location estimation within the warehouse. In this work, millimeter-wave based indoor localization system is designed that can provide the position estimates at centimeter-level accuracy and with low cost. The localization system uses multi-layer perceptron based neural network to learn the complex wireless features in an indoor environment. The localization system is further integrated with a kalman filter to provide dynamic run-time location estimations. Our proposed localization system achieves median error of 0.26m and compared to state-of-the-art wireless localization systems, achieves significant performance improvement.

Chapter 4

Adaptive Exit selection using DQN in Multi-Exit Networks

4.1 Introduction

An immediate and efficient response to a critical safety incident is crucial, as a better first response significantly impacts the safety and recovery of parties involved in the emergency. These include Emergency Medical Services (EMS), firefighters, and other law enforcement agencies. Such smart infrastructure requires technology that can provide intelligent and meaningful information regarding the situation [89]. With the recent advances in Artificial Intelligence (AI) targeting Deep Neural Networks (DNNs) for recognition, detection, and classification [52, 90–92], situation awareness systems can utilize such algorithms for their application. However, these networks consist of millions of parameters, as it has been seen that increasing the depth and complexity of DNNs has resulted in state-of-the-art performances. Table 4.1 shows the computation cost associated with commonly used DNNs. We see that the number of computations for a simpler CNN-based classification network like AlexNet is around 727M for each input image.

For the applications targeting first responder intelligence, drones or Unmanned Aerial Vehicles (UAVs) are commonly deployed to the scene of an incident and relay the critical data via different onboard sensors back to the desired agencies [18, 89, 95]. These UAVs lack the computation capability of running DNNs that are computa-

Table 4.1: computation cost (FLOPs) in commonly used DNNs

Network	Task	Input size	Computations (FLOPs)
AlexNet [51]	Classification	227x227	727M
ResNet-18 [52]	Classification	224x224	2G
SSD [93]	Detection	300 x 300	1G
Faster-RCNN [94]	Detection	600 x 850	172G

tionally expensive. Such DNNs are not practical for the internet of things (IoT) applications, and there is a need to develop DNNs that can be executed on computation and battery-constrained devices. Different techniques have been proposed and investigated in recent years.

Many approaches have been proposed in the literature to overcome the computation and energy challenges by targeting reduction in DNN parameters [2,3,96]. Work in [96] selects different DNNs based on the bandwidth requirements and minimizes the computation by selecting a network with fewer parameters when necessary. The approach is not scalable as all network models need to be stored on the hardware, increasing memory utilization. More efficient approaches proposed network partitioning, where an earlier part of the network is used to make a prediction based on the difficulty of the input image. Such networks are broadly classified as multi-exit DNNs [2,3,54,97], where intermediate classification nodes are introduced along with a final output layer. Works in [2,54] propose a threshold-based exit selection where a hand-crafted threshold is used to determine an exit. The reduction in the number of parameters and computation is achieved if the input sample takes an earlier exit, resulting in faster predictions and energy-efficient computations. While these approaches benefit the computationally constrained devices, the threshold-based exit selection strategy employed in these networks performs sub-optimally. The sub-optimal behavior is due to the hand-crafted thresholds.

Other solutions to network optimizations include network pruning, and quantization [98–100]. Approaches explored in [101] use Network Architecture Search (NAS) to optimize the DNN for the specific hardware platform. In contrast, multi-exit optimization provides several benefits as it does not focus on specific hardware platforms and can be easily generalized for any platform. In multi-exit DNN, intermediate classification exit locations are introduced, allowing the input to exit early during the inference. In our work, we design an adaptive exit strategy for multi-exit DNNs that results in faster inference time and reductions in computations.

4.2 Contribution

The main contributions of the proposed work are summarized as follows:

- **Lightweight threshold-based exit strategy:** We present a lightweight threshold based exit strategy for multi-exit DNNs utilizing the maximum probability output at exits. During inference, thresholds at each exit are compared against the probability confidence to perform the exit selection.

- **Adaptive exit strategy for optimal exit selection in multi-exit DNNs:** We propose a DQN-based adaptive exit selection in multi-exit DNNs to optimize exit selection in multi-exit DNNs. The DQN agent is first trained to learn an exit policy utilizing the history of state information. The state information is a tuple that includes accuracy, inference latency, exit decision, and image complexity. History of recent frames is used to exploit the temporal and spatial correlation in input images, resulting in a more accurate and robust exit selection.

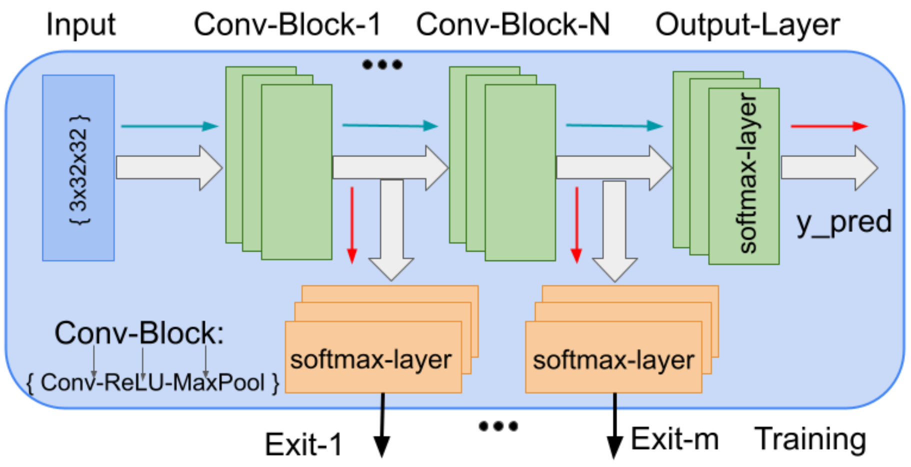
- **Image complexity aware DQN-Agent:** We propose and implement an image complexity approach to quantify the complexity associated with input images. The complexity is computed based on the minimum number of points required to represent all contours present in an input image.

- **Evaluation on wide classes of DNNs:** The proposed adaptive exit selection network is extensively evaluated using three classes of DNNs, namely, AlexNet [51], MobileNet [53], and ResNet34 [52] representing varying complexity and different CNN architectures. For evaluation, we have used a widely employed CIFAR-10 [5] image classification dataset.

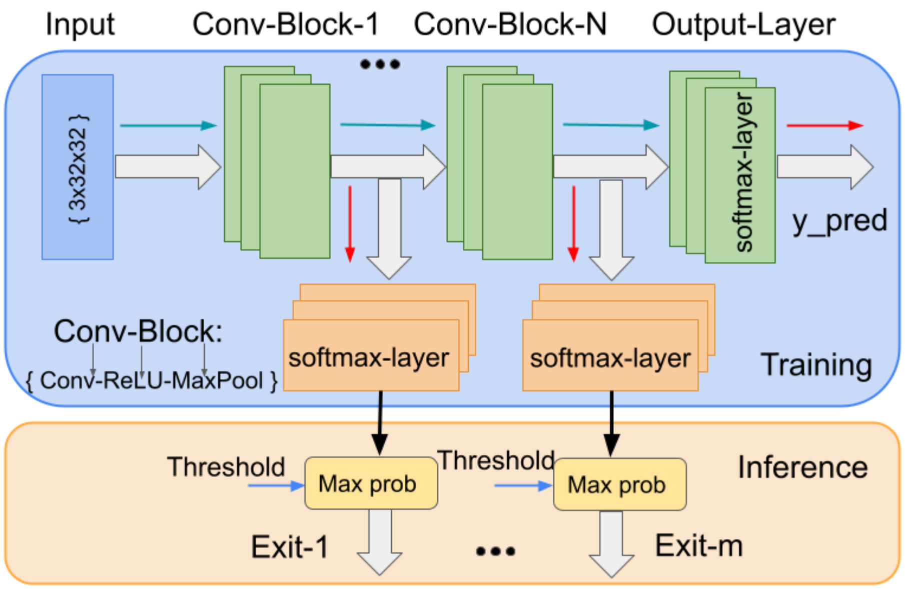
4.3 Probability Based Exit Selection in Multi-Exit DNNs

In threshold-based multi-exit DNNs, exit selection is performed by sequentially comparing threshold values at each exit. BranchyNet [2] uses the entropy of the softmax distribution as a measure of exit confidence. While EENet [54] utilizes sigmoid-based confidence at exits. In both the approaches, the computed confidence is compared against user defined threshold for performing exit selection. Our approach presents a new threshold-based exit strategy, which utilizes the maximum probability value produced at the exit as exit confidence. The final layer at each exit in multi-exit DNN uses a softmax function that outputs a vector representing the likelihood of each class and can also be interpreted as a probability distribution corresponding to all possible output classes. We formulate the confidence as the value corresponding to the maximum probability value in the distribution, which is then compared against the threshold for exit selection.

Figure 4.1a shows the network during training. We follow the training process as used in previous multi-exit DNNs [2,54]. The exit selection does not take part during the training. After the network is trained, using the test data, the thresholds are set at each of the exits. The testing process is shown in Fig. 4.1b. We sequentially iterate through all the exits, then for the exit we extract the maximum probability value. The threshold is then used to check if confidence satisfies the threshold criteria, if not, then we move to the next exit. Compared to BranchyNet [2], our approach is more lightweight as we don't perform any additional entropy computation to compute the confidence measure and directly extract the maximum confidence from the output information as provided by the multi-exit DNNs.



(a) Multi-exit DNN during the training process, the thresholds are not used at exits for exit selection.



(b) Multi-exit DNN during the testing (inference) process. The thresholds for exit selection are used at exits for exit selection using maximum probability as the confidence.

Figure 4.1: Threshold-based exit strategy in multi-exit DNN with probability based confidence

4.4 Drawbacks of Threshold-based Exit Selection

Threshold-based exit strategies suffer from three main drawbacks. Variations in threshold values affect the performance achieved by multi-exit DNNs. As the thresholds are not learned during the training, variations in threshold values significantly impact the performance of multi-exit DNNs. With hand-crafted thresholds, it becomes a difficult task to determine thresholds for each exit. For a higher selected threshold, we allow more samples to take the exit resulting in faster inference. But this can also result in degradation of classification accuracy as more samples with less confidence also take the exit.

The second drawback is the process used for selection of the threshold values. In threshold-based approaches, the test dataset is used after the training, and the thresholds at each exit are individually adjusted experimentally until the desired performance is achieved. This introduces a high bias towards the test dataset, which is not a preferred approach in machine learning.

Finally, the confidence used at an exit is another design choice that multi-exit DNNs need to make. BranchyNet [2] presented the use of entropy for the confidence measure. Work in EENet [54] proposed a sigmoid-based confidence approach, where a sigmoid function is used to compute the confidence. Early-exit BERT [3] utilized the entropy confidence similar to BranchyNet. Our threshold-based approach considers the maximum probability value in the softmax output vector as the confidence criteria. Based on our evaluation, we have seen, given the test dataset, changing the thresholds at exits results in achieving similar performance for different confidence measures.

To address the above shortcomings and drawbacks in threshold-based multi-exit DNNs, we propose designing a novel adaptive exit strategy for multi-exit DNNs using reward-based reinforcement learning. In this approach, the exit selection is considered a separate learning task, and we use a Deep Q Network to learn an optimal exit policy.

After an exit policy is learned, then during testing, the DQN agent provides the exit decision, and the multi-exit DNN takes the provided exit for the given input image. With this, we eliminate the need to use hand-crafted thresholds. In the next section, we present the design and work of a novel adaptive exit strategy in detail.

4.5 DQN-based Adaptive Exit Selection

In this section, we present our proposed adaptive exit selection strategy. Figure 4.2 illustrates the architecture for adaptive exit selection. Here, the system consists of two main components: the DQN network, DQN-Exit, which is used to learn an exit policy, and the environment, which includes a pre-trained multi-exit DNN. In our approach, the DQN is first trained to learn the adaptive exit policy by interacting with the environment in a reward-based learning manner. During training, the DQN makes an exit decision which the multi-exit DNN uses to produce an output prediction by taking the exit provided by DQN. The environment produces an output prediction, an input state, and a reward value. The DQN uses the state and the reward value to learn the exit policy. During testing, the trained DQN model provides run-time exit selection in multi-exit DNN.

Designing a DQN-based system requires several design considerations, including DQN architecture, training environment, and reward formulation. In the following sections, we will discuss each component associated with our network architecture in detail.

4.5.1 Design of Multi-Exit DNN

Our approach utilizes BranchyNet [2] to implement the baseline multi-exit DNNs. This is done as BranchyNet is one of the state-of-the-art threshold-based multi-exit approaches. An implementation using such an approach is shown in Fig. 4.3, where a multi-exit DNN consists of multiple convolution blocks with intermediate exit nodes.

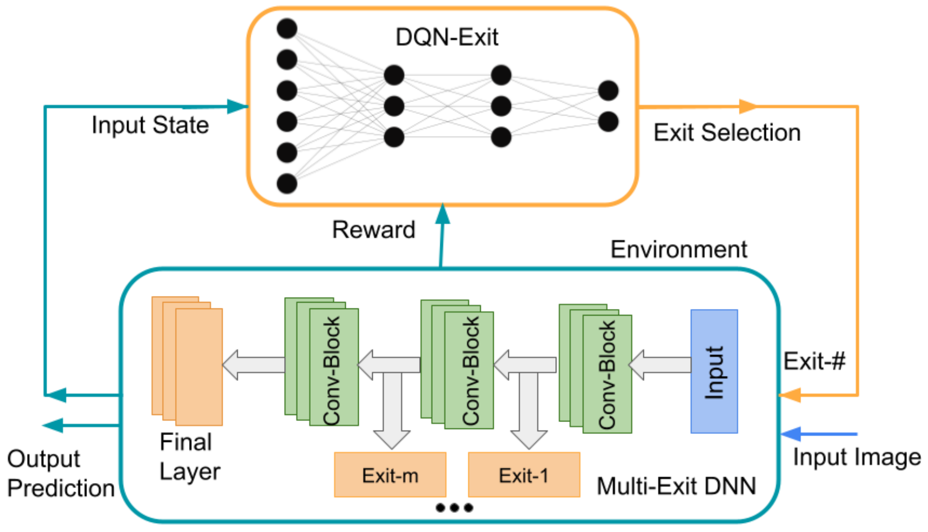


Figure 4.2: DQN based adaptive exit selection in multi-exit DNN. The thresholds are eliminated and DQN is used to learn the exit strategy

The exits have a linear layer with output neurons equal to the number of classes in the dataset and a softmax function to calculate the likelihood of each class. They can also be interpreted as a probability distribution, p_k, as shown in (4.1).

$$p_k = \frac{e^{f_k}}{\sum_j e^{f_j}} \tag{4.1}$$

Where, f_k represents the output of kth exit layer. During inference, at each exit after softmax layer, entropy is computed as a measure of exit confidence and compared against selected threshold to perform exit selection. The entropy computation is performed as shown in (4.2).

$$entropy(y) = -\sum_{C} y_c \log y_c \tag{4.2}$$

Where, y_c represent the probability corresponding to class c. The trained multi-exit DNN is used as part of the DQN environment providing the required state and reward information to the DQN-Exit Agent as shown in Fig 4.2. Next, we present the computation of the image complexity that is used to enhance the adaptive exit

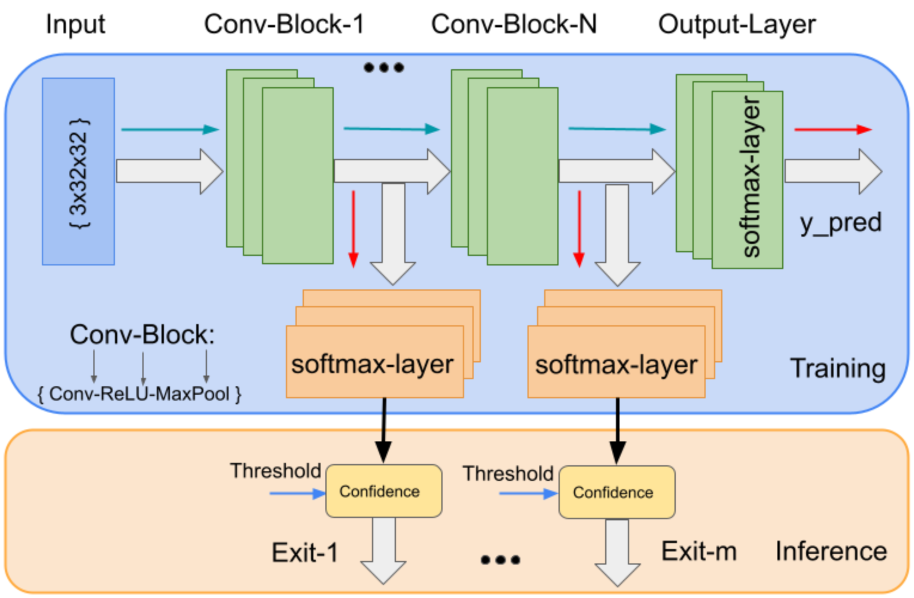


Figure 4.3: Threshold-based multi-exit DNN with threshold based exit strategy

selection strategy.

4.5.2 Image Complexity Model

Deep neural networks utilize multiple intermediate layers to learn different abstractions of image features. The initial layers learn the basic structures in an image, including simple lines and basic shapes. The middle and last layers learn more high-level semantics present in images [102]. In our approach, we learn an exit selection strategy that, based on feature complexity, selects if earlier layers are enough to provide high accuracy classification or if we need later layers to classify the image accurately. We present a contour detection-based feature extraction technique to capture the information representing complexity associated with an input image. The DQN utilizes complexity as part of its input state.

Contour extraction is performed using the contour chain approximation technique [103]. The approach is widely used in image segmentation, background extraction, and object detection applications. A contour with more unpredictability requires significantly more points than a contour with less unpredictability. The com-

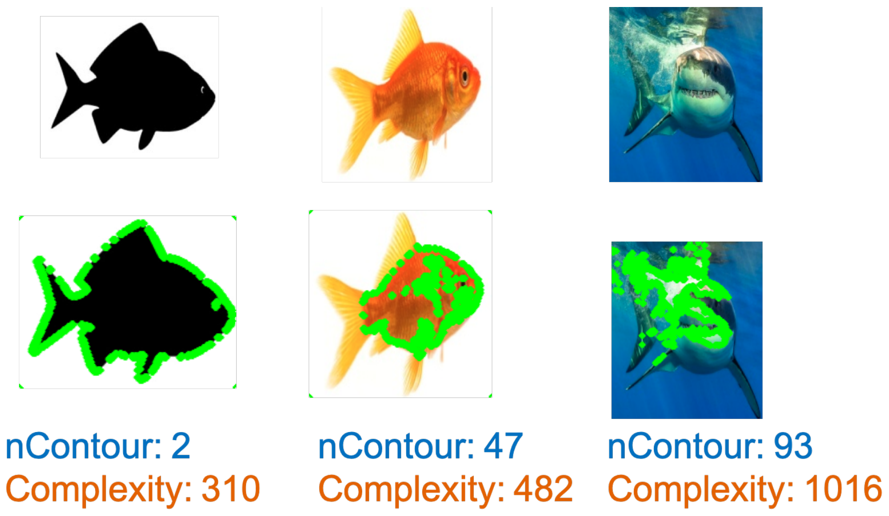


Figure 4.4: Comparing various images using complexity analysis

plexity information is extracted by first detecting all the contours present in an image and then performing the cumulative sum of the total number of points required to represent the detected contours. The extracted value is then used by the DQN-Exit, representing the complexity associated with an image. We compute the complexity as sum the all the points captured during the process and the computation can be represented as (4.3).

$$complexity = \sum_{i=1}^{nC} getPoints(Contour_i) \tag{4.3}$$

Where, nC represents total number of contours found in the image, and $getPoints$ computes the number of points required to represent a contour.

In Fig. 4.4 we perform the complexity computation on three different images representing same object class. Visually we can observe the silhouette (leftmost image) fish image will represent the least complex image compared to the other two fish images. We observe the same behavior using the complexity approach where we obtain complexity values of 310, 482, and 1016 going from leftmost image to the rightmost image. We utilize the complexity information in the form of state for the DQN-Exit agent. The objective of the agent is to learn an optimal policy resulting in reduction of computation cost by selecting an earlier exist for images that can be classified using intermediate exits. Further, to exploit the spatial and temporal correlation between sequential images, as similar images will have complexity value with fewer variations, it provides the necessary information to the DQN-Exit while learning the exit strategy.

4.5.3 DQN for Adaptive Exit Selection

We implement the exit selection using a DQN network and keep the multi-exit DNN only for image classification. Threshold approaches use multi-exit DNN to perform both image classification and exit selection. At the same time, these approaches do not provide additional features to learn an exit selection strategy effectively. This results in an inefficient exit selection in multi-exit DNNs. To overcome these shortcomings, we design a DRL-based DQN network that uses additional feature information and learns an optimal exit selection policy. With the separation of exit selection and classification networks, we maintain the high classification accuracy by only performing the classification tasks using multi-exit DNN and exit selection using the DQN agent.

Figure 4.2 illustrates the proposed system architecture for adaptive exit selection in multi-exit DNNs. The system consists of an environment model and an agent network. In our implementation, the DQN network consists of two fully connected hidden layers. The output dimension equals the total number of exits present in the multi-exit DNN. The environment is responsible for providing input state and feedback as a reward to the DQN-Exit.

DQN-Agent: The DQN network consists of two fully connected hidden layers. This architecture is considered based on the characteristics of the input features used for the DQN network. We don't use a CNN-based neural network as the DQN in our system uses a small vector of input features compared to the high feature dimension of images. In our design, the DQN agent is referred to as DQN-Exit. One of the input to the DQN-Exit during the training is the state that comprises of accuracy, inference latency, and image complexity. The DQN-Exit learns a value function approximation between the different Q-values, $Q(s_t, a_t, \theta)$, where s_t represents the state at time t, a_t is the action output, and θ represents the network parameters. The Q-function is a state-action value pair that estimates the expected sum of rewards when taking action a_t from a state s_t. In DQN-Exit, the dimension of the output layer will correspond to the action space a_t, which in our design represents the number of exits within the multi-exit DNN. The loss, as shown in (4.4), is a mean squared error function between the current Q values and the expected Q values ($\hat{Q}$). Where $L(\theta)$ represents the loss function, and N represents the batch size of image samples used during training.

$$L(\theta) = 1/N \sum_{i=N} (Q(s_t, a_t, \theta) - \hat{Q}(s_t, a_t, \theta))^2 \tag{4.4}$$

Expected Q values are computed using the Temporal Difference (TD) method, as shown in (4.5). Finally, to minimize the loss, parameter update is performed using the Gradient Descent.

$$\hat{Q} = R(s_t, a_t) + \gamma max_{a_t} Q(\hat{s}_t, \hat{a}_t, \theta) \tag{4.5}$$

The state information used in our design is a history of the previous three states represented as $s_t = [s_{t-1},\ s_{t-2},\ s_{t-3}]$, where each state at a given time consists of accuracy, inference latency, exit-decision, and complexity. The output of DQN-Exit is vector with dimension equal to number of exits in multi-exit DNN and is represented as $a_t = [exit_1,\ exit_2,\ ...,\ exit_n]$, where, $exit_n$ represents nth exit in the multi-exit DNN. Using the history-based state information for learning an adaptive exit selection strategy exploits the temporal and spatial relationship between the time-dependent input image frames making the DQN-Exit learn the relative complexity in a time-aware manner. This further optimizes the exit-selection decision-making and significantly impacts the robustness and performance of the multi-exit DNN. The idea is that images' difficulty does not vary significantly for successive input images. The exit that provides optimal performance on one of these frames should not change in closely related images. Our adaptive exit strategy learns this behavior during training, resulting in an improved system performance. Next, we discuss the formulation of reward function used to train the DQN-Exit agent.

Reward Function: The reward function is responsible for providing feedback to the DQN-Exit during the training. The reward value quantifies the action output by assigning a numerical value based on the results observed by executing the DQN-Exit action. The approach is to provide a high reward for actions resulting in improving the desired performance of the system and a lower or negative reward if the action results in performance degradation. The designed reward function focuses on three crucial aspects: 1) minimizing inference time latency, 2) achieving high classification accuracy, and 3) reducing the number of MAC operations to reduce energy. The reward computation is shown in (4.6).

$$Reward = \begin{cases} R_t, & acc \geq acc_{min} \\ R_{penalty0}, & acc < acc_{min} \end{cases} \tag{4.6}$$

Where, acc_{min}, which captures the minimum desired accuracy the agent's action should achieve; this is a common approach utilized in training DQN-Exit [58,61]. The penalty offset is the difference between the accuracy threshold and accuracy obtained from the environment. In our design, we consider the maximum accuracy achieved by the threshold-based multi-exit DNN as acc_{min}. The computed reward is given by the product of accuracy and inverse of inference time as shown in (4.7).

$$R_t = acc \cdot (1/t_{Inf}) \tag{4.7}$$

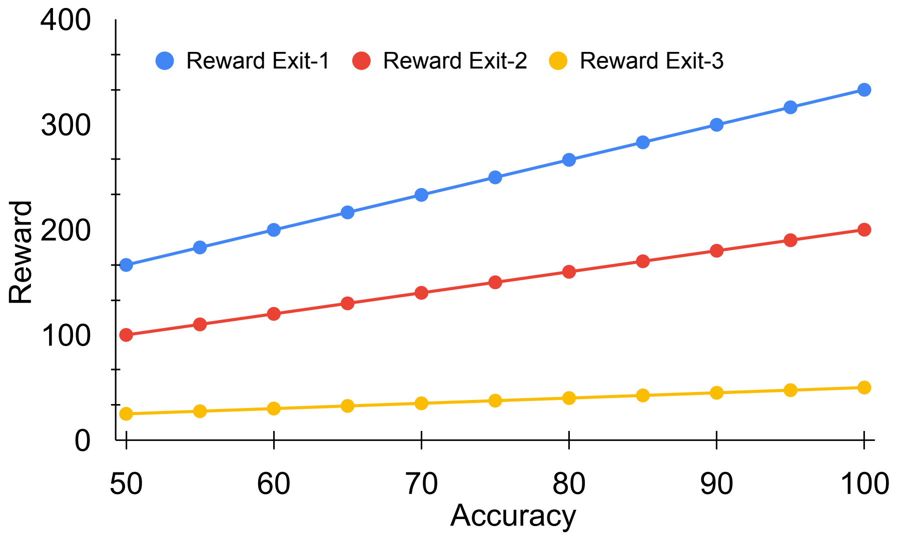


Figure 4.5: Individual reward for three exits with variation in accuracy.

Here, acc and t_{Inf} represent the accuracy and inference latency for the previous action output. In our approach, $R_{penalty0}$ is the negative reward given when accuracy produced using the DQN-Exit action falls below the minimum accuracy requirement as shown in (4.8).

$$R_{penalty0} = R_t - [(acc_{min} - acc)][(1/t_{Inf})] \tag{4.8}$$

In the reward formulation, the inverse of inference time is considered to give a relatively higher reward for the actions resulting in inference time savings. The reward penalty ensures that reward parameters do not prioritize each other, particularly as we award a higher reward for the inference latency reduction should not overshadow the contribution of accuracy. The variations of reward function based on (4.7) is shown in Fig. 4.5 where, we define three exits with different t_{Inf} and vary the accuracy. We plot a separate reward obtained for each exit with a variation in accuracy. We observe that the reward is highest for the earlier exit given a fixed accuracy as they correspond to the lower inference time.

Further, as accuracy increases, the reward difference between the exits also increases. This is desirable, as during the training process, the DQN-Exit will select the exit that maximizes the reward, i.e., for a given input sample, if an earlier exit provides the same accuracy as later exits, the DQN-Exit will learn to select the earlier exit. Based on the formulated reward function, the DQN-Exit will learn an optimal policy that will maximize the reduction in inference time while maintaining high classification accuracy.

Environment : The environment is a model or a system with which our DQN-Exit agent interacts. In our application, the environment is multi-exit DNN pre-trained for image classification. The multi-exit DNN takes two inputs, an image sample, and an exit decision provided by the DQN-Exit. As shown in Fig 4.2, the DQN-Exit provides the exit decision for the current image sample. The environment responds with a reward value and a state vector during the training. The environment model implements functionality to compute state, reward, and complexity as described in Algorithm 1. The environment supplies the state vector to DQN-Exit during training and testing. The environment is also responsible for computing the reward as described in (4.7). The environment also computes images' complexity as part of input state information.

4.5.4 Training and Testing Process for DQN

The DQN-Exit agent interacts with the environment during the training phase and learns the required adaptive exit selection policy. The training routine as shown in Algorithm 1 follows the following steps. First, using Xavier initialization, we initialize the parameters (or weights) of the DQN-Exit and the replay memory with the known capacity and set the required training episodes represented by $maxEp$. We iterate through the training dataset for each training episode, denoted by $maxIter$, generating the rewards and the next state input for the agent's network. The agent outputs an action in the form of an exit decision using the provided state and reward.

Algorithm 1 Adaptive exit selection training process

```
 0: procedure TRAIN-DQN-EXITNN($s_t, r_t, a_t$)
 1: Initialize replay memory buffer M with capacity C
 2: Initialize DNN weights, $\theta$
 3: for $episode = 1, maxEp$ do
 4:   $env.Reset()$
 5:   $reward.Reset()$
 6:   for $iteration = 1, maxIter$ do
 7:     Using $\epsilon$ either select random action or action output from DQN-Exit
 8:     Execute multi-exit DNN using $a_t$ and input image
 9:     Compute the reward, $r_t$, based on (4.7)
10:     Capture the complexity of image sample using (4.3)
11:     Formulate the next state vector, $ns_{t+1}$
12:     Store transition $s_t$, $r_t$, $done$, $ns_{t+1}$ in replay memory
13:     if done then
14:       break
15:     else
16:       Calculate the TD loss and update the agent's parameters
17:     end if
18:   end for
19: end for
```

The output is either a random action by the DQN-Exit or a Q-value estimate. This is done to introduce the idea of exploration and exploitation in DRL using an ϵ probability value.

The ϵ is a decaying probability function and is initially set to one. The decay happens with increasing iterations. During training, we employ a buffered memory called replay memory to store the current state, action, reward, termination flag, and next state during each time step. The DQN-Exit uses randomly sampled data from the replay buffer during the training. This is done to avoid correlation between consecutive samples from the environment. During the training, the objective of the DQN-Exit model is to maximize the cumulative discounted reward. After training, the trained DQN is used to make exit decisions using the held-out test samples.

During testing, the pre-trained DQN-Exit agent outputs an exit decision which the multi-exit DNN utilizes in the environment. Based on the accuracy, inference

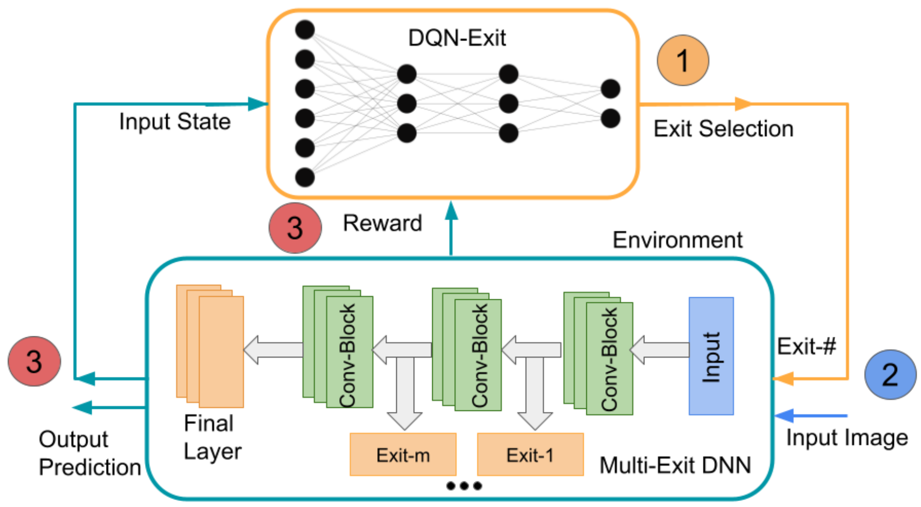


Figure 4.6: DQN Training is a three-step process. In Step 1, the DQN provides an exit number for the pre-trained multi-exit DNN. In Step 2, the multi-exit DNN utilizes the exit number for the input image. In Step 3, the multi-exit DNN outputs a prediction along with the state and reward for the DQN. The whole process is repeated over a defined number of episodes for the DQN to learn the exit strategy. During the testing, all steps are performed except for the reward, as the DQN is not updating its parameters so we don't need a reward during the testing or inference process.

time, previous exit, and image complexity next state is generated, which is sent to the DQN-Exit for the next exit decision. It is to note that during testing, DQN parameters are not updated, and parameters are only learned during training. Figure 4.6 also depicts the interaction of the DQN-Exit with the multi-exit DNN.

4.6 Performance Benefits with Adaptive Exit Strategy

This section compares the performance benefits achieved using the proposed DQN-based adaptive exit selection over threshold-based approaches. For this, we evaluate the cost for multi-exit DNNs in terms of Multiply and Accumulate (MAC) computations and inference time latency. Figure 4.7 illustrates the operations using the threshold-based and DQN-based exit strategy using a multi-exit DNN with two intermediate exits. With the threshold exit strategy taking an exit also results in an additional cost if there are any previous not taken exits. Whereas in the adaptive ap-

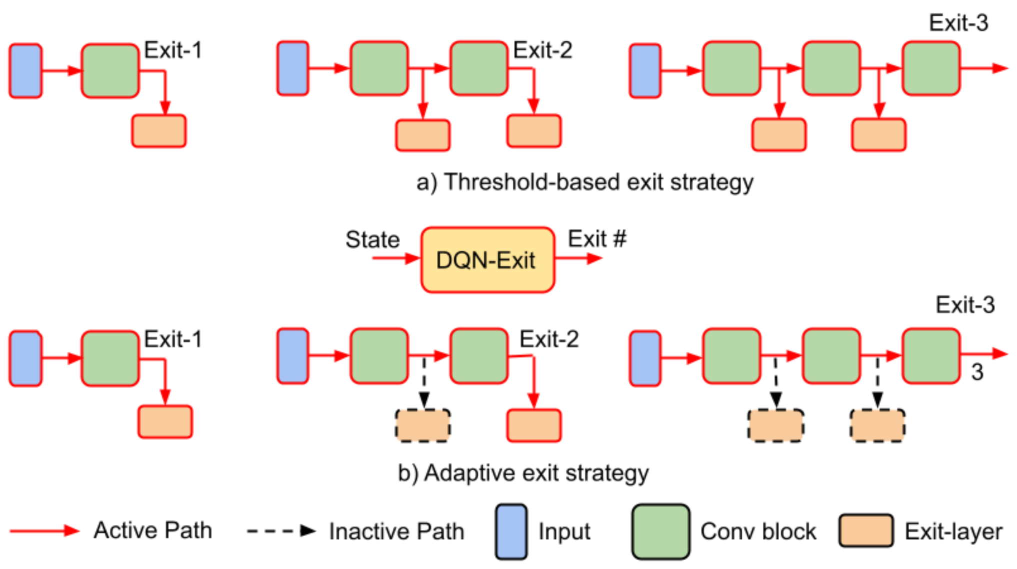


Figure 4.7: Computation and inference time cost behavior in multi-exit DNNs. a) shows the threshold-based exit strategy [2] in multi-exit DNN with three exits. For each later exit, we incur costs associated with all previous exits. b) shows the proposed DQN-based adaptive exit strategy. For each later exit, the previous exit paths are inactive achieving a reduction in computations and inference cost.

proach, the exit is provided for the multi-exit DNN. As a result, the multi-exit DNN takes the provided exit without incurring additional costs for any previous exits. In Fig 4.8 we show an example where Exit-2 is selected for both threshold and adaptive approaches. We see with the threshold-based strategy, the cost associated with the Exit-1 will also be included. But with the adaptive strategy, input directly takes the path from the input layer to the Exit-2 and keeps the Exit-1 path inactive. From this, we observe that if the same exit is taken using the threshold-based and the adaptive exit strategy, the total cost associated with the adaptive multi-exit DNN will be lower compared to the threshold-based strategy.

Based on the above observation, we formulate the cost associated with threshold based and DQN-based adaptive exit selection as shown in (4.9), and (4.10).

$$Cost_{Threshold}^{Exit_N} = Cost_{Conv}^{\{1:N\}} + Cost_{Exit}^{\{1:N\}} \tag{4.9}$$

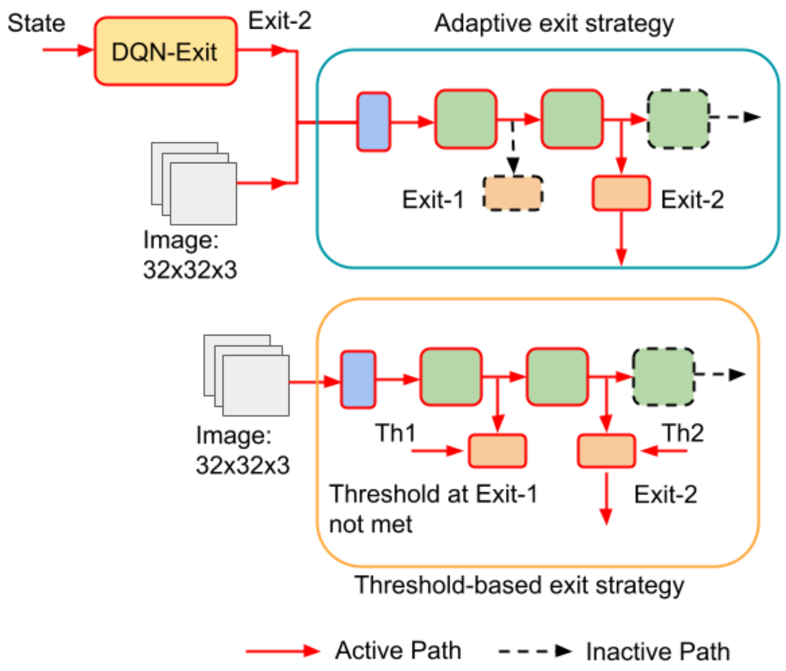


Figure 4.8: Inference comparison between adaptive exit selection and threshold-based exit selection for the for same taken exit in a multi-exit DNN

$$Cost_{adaptive}^{Exit_N} = Cost_{Conv}^{\{1:N\}} + Cost_{Exit}^{\{N\}} + Cost_{DQN} \tag{4.10}$$

Where $Cost_{Conv}^{\{1:N\}}$ represents the cost associated with the conv-blocks up to the selected exit, Exit-N, and $Cost_{Exit}^{\{1:N\}}$ represents the cumulative cost associated with the exits starting from the first to the selected exit in the multi-exit DNN. DQN-Exit cost is represented by $Cost_{DQN}$ in (4.10).

The adaptive exit strategy formulates the exit selection as a learning task by utilizing a DQN-Exit to learn the optimal exit selection policy. The introduction of DQN-Exit adds additional cost overhead to our adaptive exit strategy. But from our analysis and the design of the DQN-Exit agent, we observe that the total MAC cost of the DQN-Exit agent is less than the MAC cost associated with an individual exit node. As a result of which, for every exit, except for the first, $Cost_{adaptive}^{Exit_N}$ will be less than $Cost_{Threshold}^{Exit_N}$.

For the first exit, the total cost overhead using the adaptive exit strategy will be

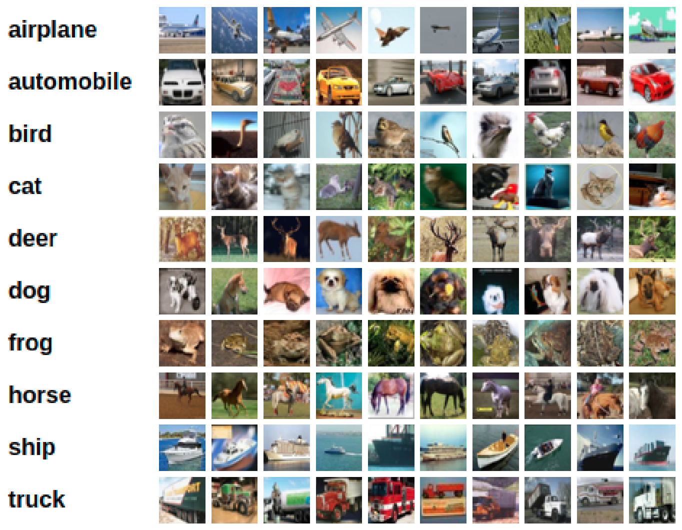


Figure 4.9: CIFAR10 image classification dataset [4]

higher than the threshold-based exit strategy due to the DQN-Exit cost overhead. The corner case, where most samples take the first exit in both threshold-based and adaptive approaches, will result in a higher cost overhead using adaptive exit selection. But from our evaluation in Section 4.7.6, we observe that the adaptive exit strategy results in a higher percentage of samples taking an earlier exit compared to threshold-based approaches. As a result, the overall cost overhead using the DQN-based adaptive exit selection is lower than threshold-based exit selection.

4.7 Experimental Analysis

In our evaluation, we have considered three different DNNs, namely, AlexNet [51], MobileNet [53], and ResNet34 [52]. These selected DNNs represent varying complexity and different architectures, providing an accurate and realistic system evaluation. We introduce additional convolutional layers in a CNN based AlexNet [51]. The MobileNet uses depth-wise and point-wise convolutions, and ResNet is based on residual layers. This shows a wide applicability of our adaptive exit strategy. Using the three DNNs, we implement the baseline multi-exit DNN by introducing intermediate exit locations. The three baseline multi-exit DNNs are referred to as Exit-AlexNet,

Table 4.2: Total computations (MAC) for different exit positions

Network	Exit-1	Exit-2	Exit-3	Exit-4
Exit-AlexNet	43.5M	60.5M	62.7M	-
Exit-MobileNet	2.5M	10M	11.6M	-
Exit-ResNet	7.2M	11.9M	16.7M	75M

Exit-MobileNet, and Exit-ResNet. We refer to our DQN based adaptive multi-exit DNN networks as DQN-ExitAlexNet, DQN-ExitMobileNet, and DQN-ExitResNet, respectively. Our design of AlexNet consists of five convolutional (Conv) blocks and two fully connected (FC) layers at the output. We augment the AlexNet with two exits, after the first and third Conv blocks. In Exit-ResNet, we introduce three intermediate exits after the first, third, and fifth Conv block. Exit-MobileNet consists of three Conv blocks with multiple depth-wise and point-wise convolutions. We add two exits, one after the first block and the second exit after the second block. The threshold-based multi-exit DNNs are trained using the approach presented in [2]. We modify the framework presented in [54] for designing multi-exit DNNs using Pytorch.

The total computation cost for all multi-exit DNNs is shown in Table 4.2. We also utilize a CPU and a GPU-based two different computation hardware to perform the testing. For evaluation, we compare our adaptive selection networks with threshold-based multi-exit DNNs along with traditional end-to-end DNNs. We use Pytorch to model multi-exit DNNs and DQN agent networks. We use CIFAR-10 [5] dataset in our evaluation. CIFAR-10 consists of 60,000 32x32 RGB images of 10 different classes. There are 50,000 training images and 10,000 test images. From the 10,000 test images, we divide it into two splits of 5000 each. The first split is used to train the DQN network, and the second is used to evaluate the proposed adaptive exit system.

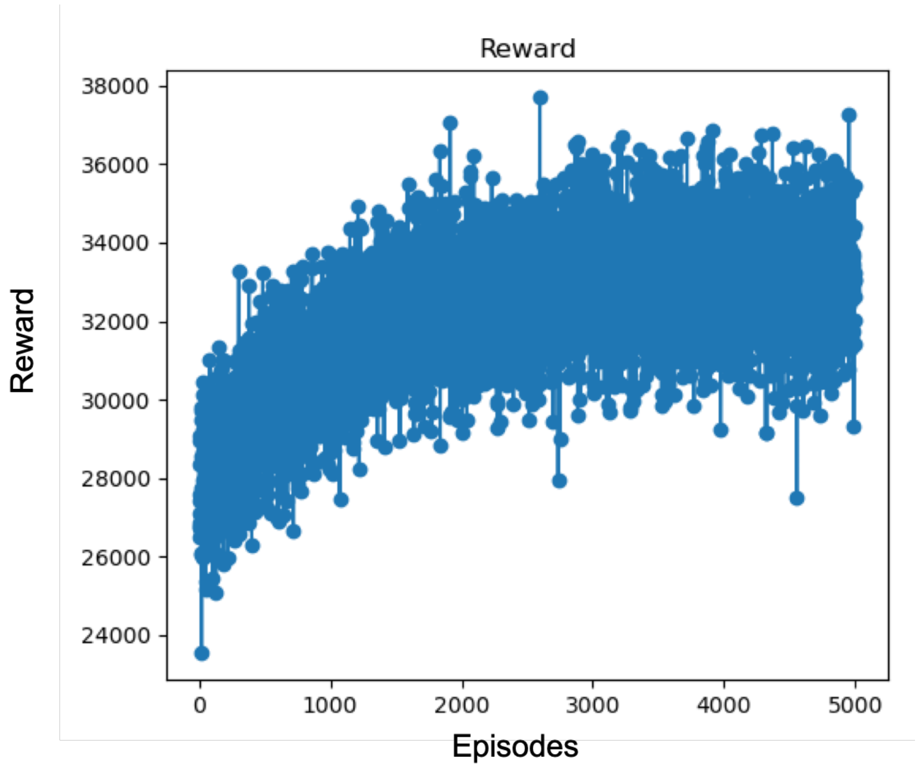


Figure 4.10: Reward received during the training of DQN-ExitAlexNet

4.7.1 Training Exit Selection Policy using DQN

We follow the training process as described in Section 4.5.4 to train the DQN-Exit agent. The training progress in DRL is monitored by observing the reward received by the DQN-Exit agent over the defined training episodes. For Exit-AlexNet as an environment model with three exits, we set the number of training episodes to 5000. For each episode, the multi-exit DNN uses the training dataset to output the next state, reward, and output prediction for the images. For exploration and exploitation during the training, epsilon decay was set to a value of 2000 and we record the loss and the reward obtained for each episode in a separate variable. The epsilon value is initialized to one at the start of the training. A higher value of epsilon indicates we will take more random actions instead of the action output provided by the DQN agent. This means we explore by doing random action selection. The epsilon decreases with the number of training steps and with lower epsilon values we utilize action output

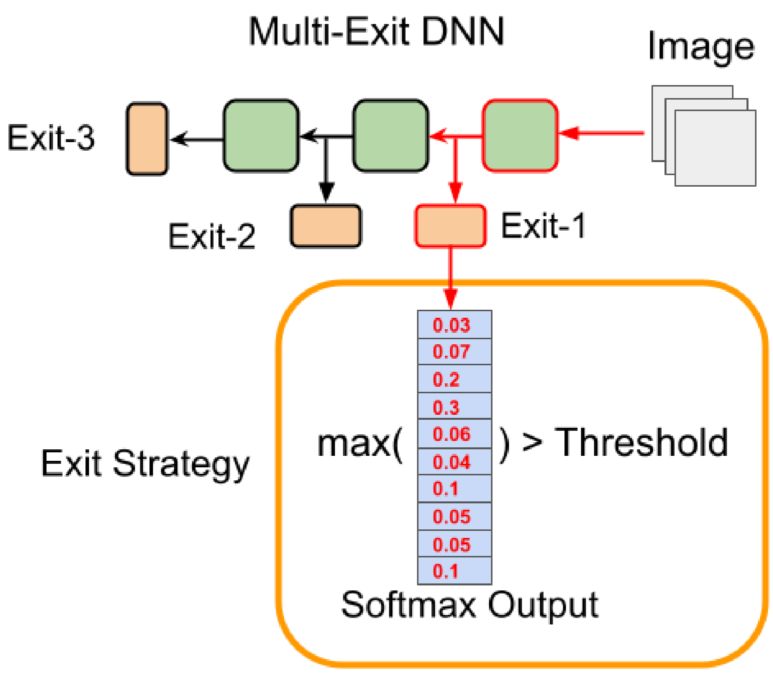


Figure 4.11: Using maximum probability as threshold criteria for multi-exit DNN

from the DQN. From Fig. 4.10 we see the reward obtained during the training of DQN-ExitAlexNet. The reward shows an increasing cumulative value with the number of training iterations and converges towards the end.

4.7.2 Threshold-based Exit Strategy in Multi-Exit DNN

Here, we implement the proposed probability-based exit strategy using three multi-exit DNNs. We compare the performance with two state-of-the-art threshold-based exit approaches, BranchyNet [2], and EENet [54]. BranchyNet uses entropy as threshold and EENet uses sigmoid as threshold for exit selection. In Table 4.3, we compare the proposed threshold-based exit strategy with an entropy-based exit strategy [2] and a sigmoid-based exit strategy [54]. In our evaluation, we have used three multi-exit DNNs, namely, Exit-AlexNet, Exit-MobileNet, and Exit-ResNet. We observe with the proposed probability-based exit strategy, all the three multi-exit DNNs achieve the highest classification accuracy with 81.03% for Exit-AlexNet, 91% for Exit-MobileNet,

Table 4.3: Performance comparison with different threshold-based exit strategies

Network	Threshold	Accuracy (%)	Inf. Time CPU (ms)	Exit (%)
Exit-AlexNet	Entropy [2]	80.06	3.54	[69, 31, 0]
	Sigmoid [54]	80.00	3.66	[73, 12, 15]
	Probability	**81.03**	**3.59**	**[68, 28, 4]**
Exit-MobileNet	Entropy [2]	90.10	3.80	[0, 96, 4]
	Sigmoid [54]	89.00	3.70	[0, 100, 0]
	Probability	**91.00**	**3.76**	**[0, 97, 3]**
Exit-ResNet	Entropy [2]	88.00	2.32	[0, 0, 100, 0]
	Sigmoid [54]	88.00	2.32	[0, 0, 100, 0]
	Probability	**89.05**	**2.63**	**[0, 0, 95, 5]**

and 89.05% for Exit-ResNet.

In Exit-AlexNet, with probability as a threshold, we see a 1.9% reduction in inference latency compared to the sigmoid-based approach. This is seen due to the higher percentage of samples taking earlier exits with a probability-based exit strategy. We make a similar observation in Exit-MobileNet, where we achieve a faster inference time with probability than the entropy-based exit strategy due to a higher percentage of samples taking earlier exits. While in Exit-ResNet, with our approach, we have a higher number of samples taking the later exit, which results in a higher inference time. With all the three threshold-based exit approaches, the performance depends on the thresholds selected at each of the exit output. Changing the threshold value will result in changing the exit distribution. Intuitively, when we decrease the threshold at an exit, we allow for more samples to take the exit, and if we increase the threshold value, fewer samples will take the exit. Figure 4.12 illustrates this variation, where we change the threshold for Exit-1 in Exit-AlexNet and observe the impact on network accuracy. From Fig. 4.12 we can see that with an increasing threshold value, the input is required to have high classification confidence to take the exit. With a higher confidence requirement, fewer samples will take the exit and move towards

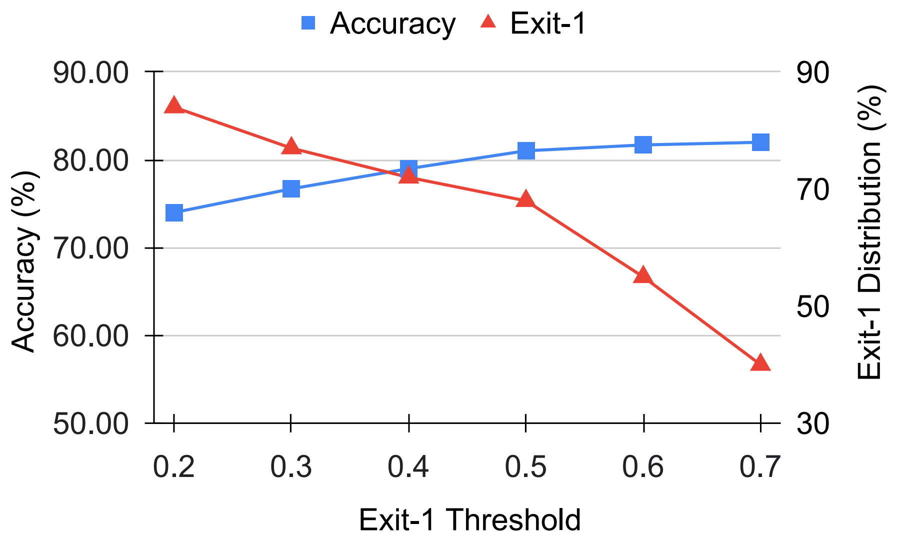


Figure 4.12: Exit-AlexNet accuracy with variation in Exit-1 threshold using probability based threshold-based exit strategy

the next later exit in the network, which results in increased accuracy. With a lower threshold value, more samples with low confidence will take the exit. With a higher number of samples with low confidence taking the exit, the probability of incorrectly classifying an object increases, resulting in lower classification accuracy.

Based on this observation, we also note that by changing the threshold values, similar performance benefits can be achieved using any of the mentioned threshold-based exit strategies. This is due to the non-optimal threshold selection process where the thresholds are set manually to fit the test dataset introducing a high dataset bias in all threshold-based exit approaches.

4.7.3 Classification Accuracy with DQN based Adaptive Exit Strategy

We compare the performance of DQN based adaptive exit selection with baseline threshold exit criterion and traditional end-to-end DNNs. The performance is measured by evaluating inference time, classification accuracy, and total MAC operations. The threshold-based multi-exit DNNs considered in our evaluation are trained using the training approach described in Section 4.5.4. From Table 4.4 we see that end-to-

Table 4.4: Performance comparison of different DNNs with DQN-based exit selection and threshold-based exit-selection on CIFAR-10 dataset

Network	Accuracy (%)	Inf. Time GPU (ms)	Inf. Time CPU (ms)	Exit (%)
AlexNet	82.50	0.050	5.00	[0, 0, 100]
Exit-AlexNet [2]	80.06	0.035	3.54	[69, 31, 0]
DQN-ExitAlexNet	**81.30**	**0.034**	**3.30**	**[80, 17.90, 2.10]**
MobileNet	92.00	0.173	5.70	[0, 0, 100]
Exit-MobileNet [2]	90.10	0.120	3.80	[0, 96, 4]
DQN-ExitMobileNet	**90.78**	**0.100**	**3.60**	**[3, 97, 0]**
ResNet34	90.80	0.200	9.04	[0, 0, 0, 100]
Exit-ResNet [2]	88.00	0.070	2.32	[0, 0, 100, 0]
DQN-ExitResNet	**89.20**	**0.063**	**1.76**	**[0, 70, 30, 0]**

end DNNs obtain the highest classification accuracy for all the three classes of DNNs. As all the features present in the deep layers are used for the output prediction, the probability of samples getting classified correctly increases. But this also results in a high computation cost and inference time for the input samples.

In multi-exit DNNs, inputs can use earlier exits to significantly improve the inference time and computations based on exit strategy. In DQN-based adaptive multi-exit DNNs, DQN-ExitAlexNet, DQN-ExitMobileNet, and DQN-ExitResNet, we achieve a high classification accuracy of 81.3%, 90.78%, and 89.2% respectively. Although in adaptive multi-exit DNNs, we observe a slight reduction in accuracy compared to end-to-end DNNs. Compared to threshold-based multi-exit DNNs, all three adaptive multi-exit DNNs show improvement in classification accuracy. In particular, we see from Table 4.4 a 1.5% increase in accuracy with DQN-ExitAlexNet, a 0.75% with DQN-ExitMobileNet, and a 1.36% increase with DQN-ExitResNet. While this is a slight improvement in accuracy, it shows that adaptive multi-exit DNNs are more capable of achieving accuracy closer to end-to-end DNNs than threshold-based exit selection for a significant reduction in inference time which is discussed next.

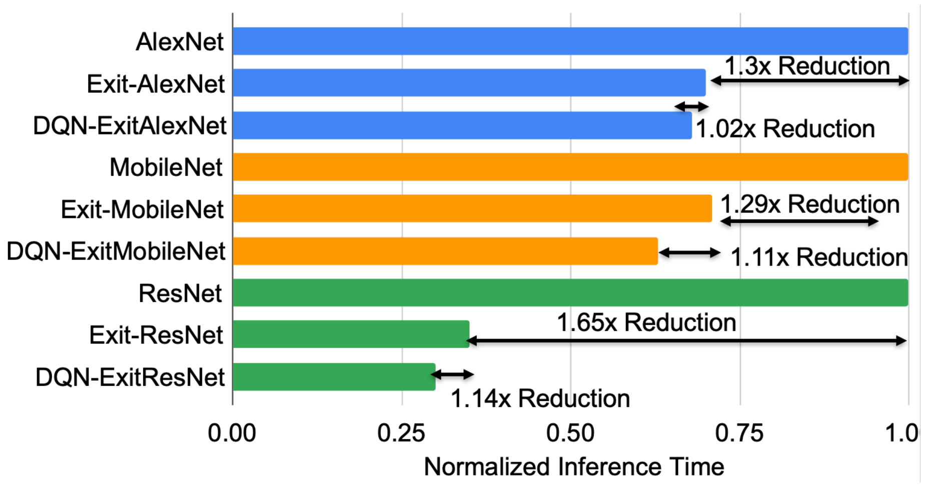


Figure 4.13: Normalized latency reductions achieved in DQN multi-exit DNNs compared to threshold-based multi-exit DNNs and traditional DNNs.

4.7.4 Exit Distribution and Inference Time Analysis

Inference time latency is a critical performance metric that we analyze. Inference latency is measured as the total time required to output a prediction after input is fed to the multi-exit DNNs. The inference time also governs the rate of real-time operation, and achieving a reduction in this will significantly impact the network's performance. In our analysis, the inference is normalized with-respect-to traditional end-to-end DNNs and we have considered the CPU based system for the analysis. The inference time represents the total time including the execution time of DQN-Exit agent, multi-exit DNN and the complexity computation. For complexity the time required on CPU system is recorded to be 0.36ms. From Fig. 4.13 we see that DQN-ExitAlexNet, DQN-ExitMobileNet, and DQN-ExitResNet, all the three adaptive exit DNNs achieve 32%, 37%, and 70% reduction in inference latency compared to AlexNet, MobileNet, and ResNet, respectively. Similarly, compared to traditional DNNs, all threshold-based exit selection networks Exit-AlexNet, Exit-MobileNet, and Exit-ResNet also achieve 30%, 29%, and 65% reduction in inference latency. From this, we observe that we achieve a significant reduction in total inference time latency for both threshold-based and adaptive exit selection approaches. This observation,

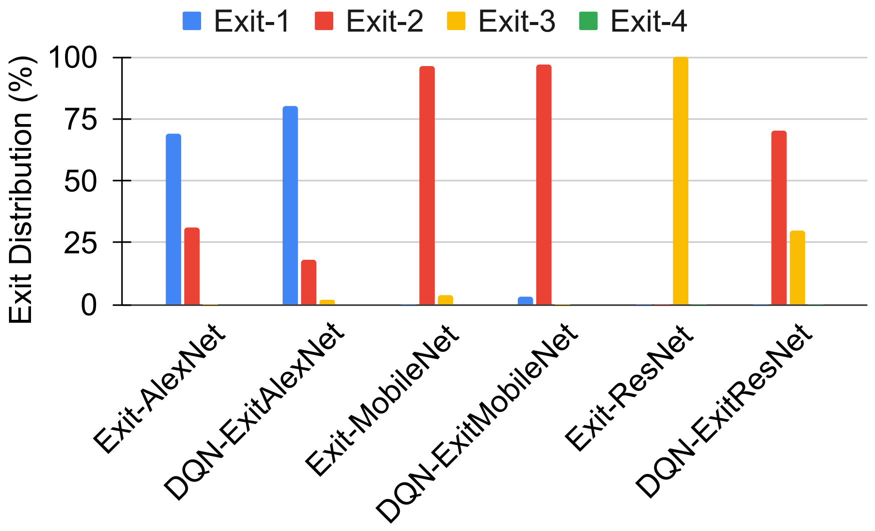


Figure 4.14: Exit distribution in threshold-based and adaptive exit selection

coupled with our previous analysis of classification accuracy, clearly justifies the significant benefits achieved by introducing intermediate exits in DNNs.

Comparing the threshold-based exit strategy with our adaptive DQN based exit strategy, DQN-ExitAlexNet, DQN-ExitMobileNet, and DQN-ExitResNet achieve 2.8%, 11.2%, and 14.2% lower inference latency compared to Exit-AlexNet, Exit-MobileNet, and Exit-ResNet respectively. This reduction in inference time can be seen due to the difference in exit distribution between the threshold-based and the adaptive exit strategy. The adaptive exit selection optimizes the exit based on a history of state information, including accuracy, inference time, previous exits, and image complexity. The state provides a deeper understanding of exit behavior to the DQN network, resulting in a more optimal exit strategy that achieves high classification accuracy and reduces inference latency. Figure 4.14 illustrates the exit distribution for the three multi-exit DNNs with threshold-based and the proposed adaptive exit strategy. Compared to the threshold-based exit selection, we observe that with the adaptive approach, all the three networks result in higher distribution input samples taking an earlier exit.

In DQN-ExitAlexNet, 80% of samples take the first exit and only 17.9%, and 2.1% take second and third exit. Whereas, in Exit-AlexNet, 31% samples take the second exit, increasing the inference time and computations. The exit selection in Exit-AlexNet happens sequentially, and each exit is checked for threshold until the threshold requirement is met, introducing extra timing overhead. While in the adaptive selection, the DQN-Exit outputs an exit decision for the incoming input image. The multi-exit DNN takes the provided exit without going to each exit sequentially and helps in reducing the additional sequential timing overhead even if the same exit decision is made in both networks. Similarly, we see in Exit-ResNet all the samples take Exit-3 whereas, in DQN-ExitResNet, the exits are distributed to maintain higher classification accuracy and faster inference. In DQN-ExitMobileNet, 1.4% more samples take Exit-2, compared to Exit-MobileNet and last exit is not taken for any of the input samples. The analysis shows that all three adaptive multi-exit DNNs reduce inference latency due to the intelligent exit selection strategy resulting in an optimal exit distribution for achieving faster inference rate while achieving higher accuracy compared to threshold-based exit selection.

We also evaluate the inference time for the multi-exit DNNs using a GPU based hardware. Table 4.4 shows the inference time for a GPU based system. Compared to CPU, we see considerable decrease in inference time due parallel acceleration provided by GPU architecture for large matrix multiplications. We observe with the change in hardware the relative inference performance between the multi-exit DNNs remains same as previously discussed for CPU based system.

4.7.5 Study of Complexity with Exit Selection

Image complexity is computed as described in Section 4.5.2, where for an image we extract total number of points required to construct all the contours. Figure 4.15 and 4.16 shows the complexity values on CIFAR-10 images. We observe the complexity

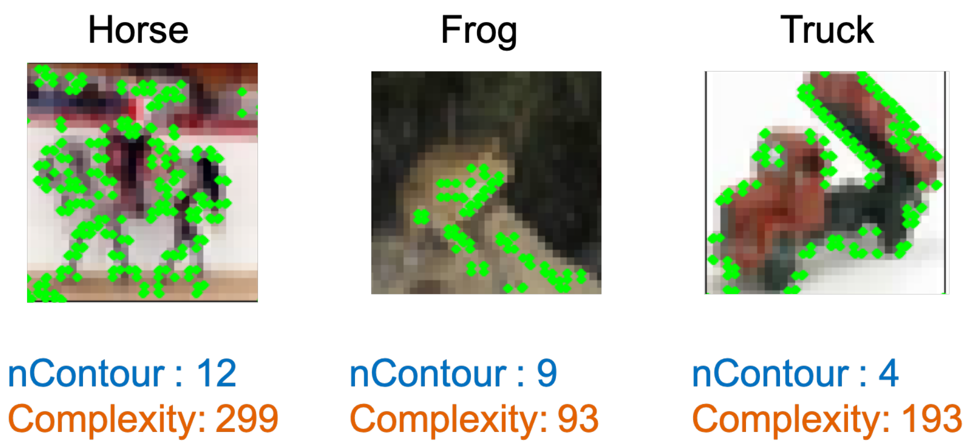


Figure 4.15: Comparing various CIFAR-10 classes using complexity analysis [5]

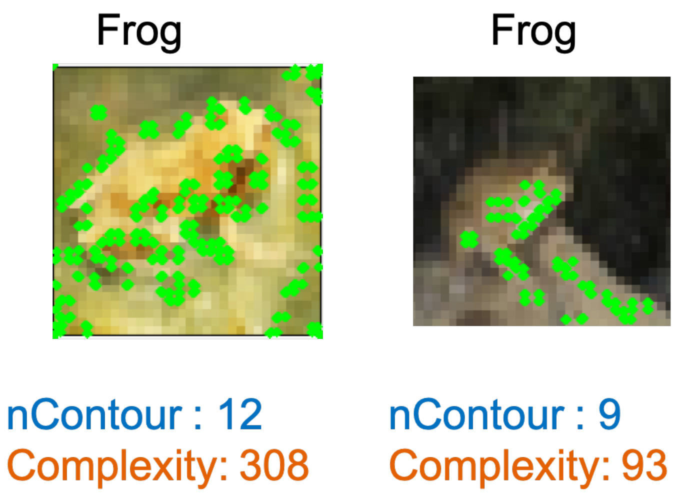


Figure 4.16: Complexity values for same class objects in CIFAR-10

varies between objects of different classes as shown in Fig. 4.15. Images corresponding to same class also show variation in computed complexity as shown in Fig. 4.16. Samples with more complex number of contours will result in a higher complexity compared to image samples with fewer number of contours. We use the complexity information to represent the difficulty associated with an image and provide that as an input feature to the DQN-Exit while learning the exit strategy. While complexity is only providing an approximation of image feature information to the DQN-Exit agent, but it helps the DQN to learn a distribution that can relate the complexity

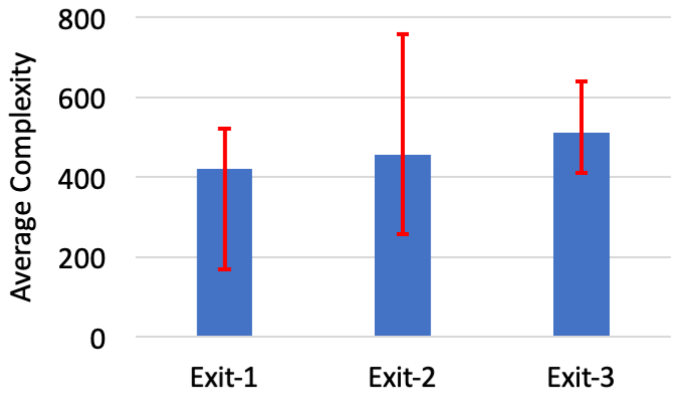


Figure 4.17: Average complexity value associated with different exits in DQN-ExitAlexNet

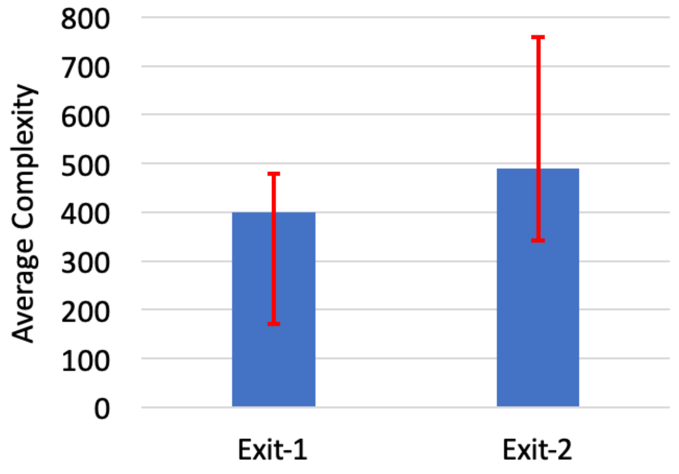


Figure 4.18: Average complexity value associated with different exits in DQN-ExitMobileNet

associated with images with exit selection.

Figure 4.17 illustrates the plot with the average complexity values and minimum and maximum variation of complexity from the mean in DQN-ExitAlexNet. We observe the higher average complexity for the last exit, Exit-3, and the low variation from the average value. For the first exit, Exit-1, we obtain a higher variation in the minimum value. For Exit-2, we observe higher variations in both maximum and the minimum values along with overlap of variations with Exit-3. The high variation and overlap in the complexity are observed due to images with similar complexity values taking different exits. As the computed complexity is an approximation provided

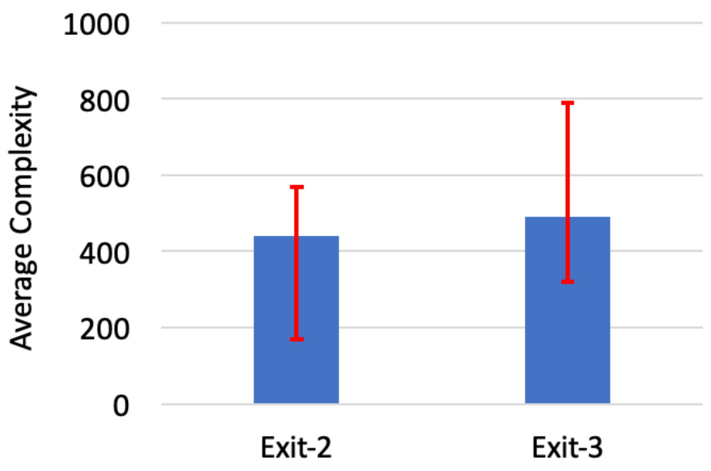


Figure 4.19: Average complexity value associated with different exits in DQN-ExitResNet

to the DQN agent, there can be instances where a lower complexity image may require a later exit for correct classification. We make similar observation in DQN-ExitMobileNet, as shown in In Fig. 4.18, we see a lower average complexity and low variation for the maximum value for Exit-1. For Exit-2, we get a higher variation in the minimum and the maximum values. This is due to the more variations in complexity values for samples taking Exit-2 compared to Exit-1.

In DQN-ExitResNet, Exit-1 again shows high minimum value variation and Exit-2 shows high variation in the maximum value, as shown in Fig. 4.19. From these results, we see the mean complexity value follows an increasing behavior with the later exits for all the three networks, which indicates high complexity images use later exits. But we also observe variations from the mean value among the exits, which shows that images with high complexity also benefit by taking an earlier exit. Similarly, less complex images can also benefit more by taking a later exit. The analysis shows while the DQN uses complexity to learn an optimal exit strategy, exits are not strongly correlated to the complexity values. Therefore we cannot use complexity directly as a threshold to perform the exit selection and need a more optimal approach to learn exit selection in multi-exit DNNs.

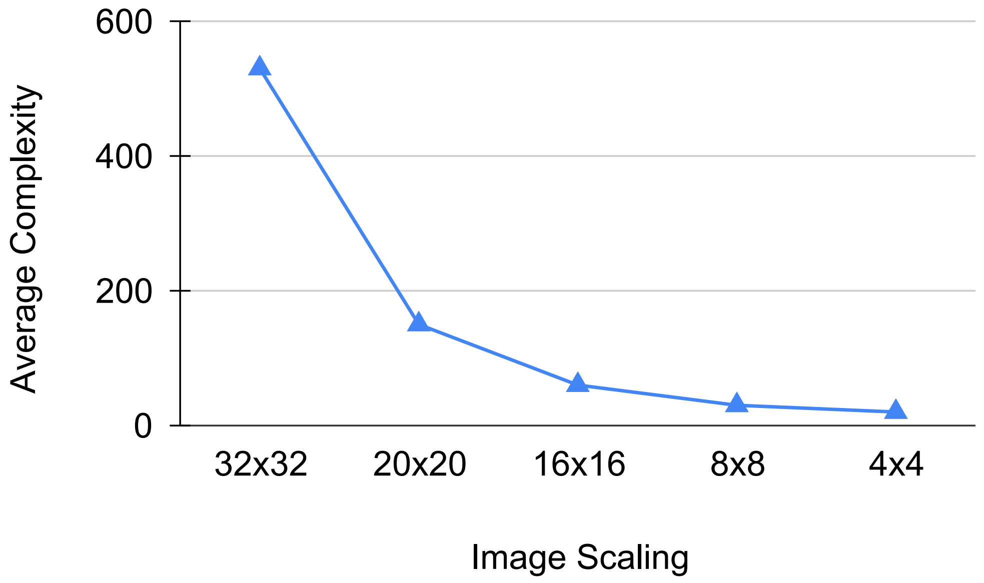


Figure 4.20: Variation of complexity with image scaling on CIFAR-10 [4] test dataset

4.7.5.1 Impact of image scaling on complexity

Here we study the impact of image scaling on computed complexity. It is to note that image scaling is not part of the adaptive exit selection system as the input dimension of multi-exit DNN will be fixed. Here we want to study how complexity may vary if the dimension of the given image dataset is scaled. Figure 4.20 illustrates the average complexity of the CIFAR-10 test dataset when the images are scaled down. We see the computed complexity decreases non-linearly as we scale down the images. When we reach the dimension of 8x8 and below, we observe a significant drop in complexity. This is seen as with image scaling, we lose a lot of pixel-based spatial information present in the images, which results in a reduction of the number of detected contours that segments multiple objects. Authors in [104] presented a similar study where they analyzed the impact of variations in the image resolution on object segmentation performance. Images with higher resolution were found to detect more contours than images with lower resolution due to more pixel-based information at higher resolution. Work in [105] studied the effect on contour boundaries as variation in segmentation scale. The results presented showed the number of detected objects in the image decreases non-linearly with increasing scale. The observations in [104, 105] follow our analysis where the computed complexity decreases with scaling due to a reduction in the number of contours detected.

4.7.6 Computation Cost Analysis

DQN introduces additional computational overhead for each exit decision. The implemented DQN utilizes 100 MAC computations for DQN-ExitAlexNet and DQN-ExitMobileNet and 115 MAC operations in DQN-ExitResNet. However, even with added complexity with DQN, our approach saves significant MAC operations in multi-exit DNNs. In DQN-ExitAlexNet, each intermediate exit consumes 960 MAC operations. We have 43M MAC operations before the first exit, 17M MAC between the first and second exit, and 2M MAC for the final exit. Table 4.5 shows the test image distribution between DQN-ExitAlexNet and Exit-AlexNet. The computations represent the difference in total MAC computations between DQN-ExitAlexNet and Exit-AlexNet corresponding to an exit. The positive values indicate that DQN-ExitAlexNet requires higher computations, and the negative value indicates the computations saved by DQN-ExitAlexNet. After all test images, we see that the DQN-ExitAlexNet achieves a 3.4% reduction in MAC computations compared to Exit-AlexNet.

Further, compared to the traditional AlexNet, DQN-ExitAlexNet achieves a significant 25% reduction in computations. The computation savings will be higher for deeper DQN-ExitResNet, as the number of computations increases between the exits. The reduction in computations for DQN-based adaptive exit selection is due to a higher percentage of samples taking earlier exits relative to the threshold-based exit selection, which is shown in Table 4.4. Further, using Table 4.2 in DQN-ExitResNet we achieve 20.1% reduction in MAC operations compared to threshold-based Exit-

Table 4.5: Computation cost benefits in adaptive DQN-ExitAlexNet with respect to threshold-based Exit-AlexNet

#Exit	Image Samples per exit		Computations [a]
	Exit-AlexNet	DQN-ExitAlexNet	(MAC)
Exit-1	3,500	4,000	+21,750M
Exit-2	1,500	900	-36,300M
Exit-3	-	100	+6,270M
Savings			**-8,280M**

[a]Computation overhead for DQN-ExitAlexNet relative to Exit-AlexNet

ResNet and 82% compared to ResNet34.

4.7.7 Adaptive Exit Selection with Sequential Input Images

The DQN based adaptive exit selection exploits the spatial and temporal dependence in input frames resulting in more adaptive exit selection. To create frames representing a video data, we pass the images through an image augmentation pipeline. The pipeline includes random image rotation, and pixel cropping operation on each of the selected image frames to represent the effects of random sensor rotation or zooming. Multiple such clips generated from individual images are appended together in time to represent video data with a scene change. We adopt this approach for synthetic video frame generation instead of using a video dataset to avoid the added effort of data labeling and re-training of multi-exit DNN and the DQN. However, our adaptive exit selection approach is readily applicable to any video classification dataset. We use the synthetic video clip obtained using the process described above to test the adaptive exit selection in our adaptive exit approach. In Fig. 4.21, the complexity plot shows the complexity values on the synthetic frames. The complexity values remain in the same range as the original image after we perform the augmentation. With this, we introduce the variation in the scene by using the frame of different complexity in a single video.

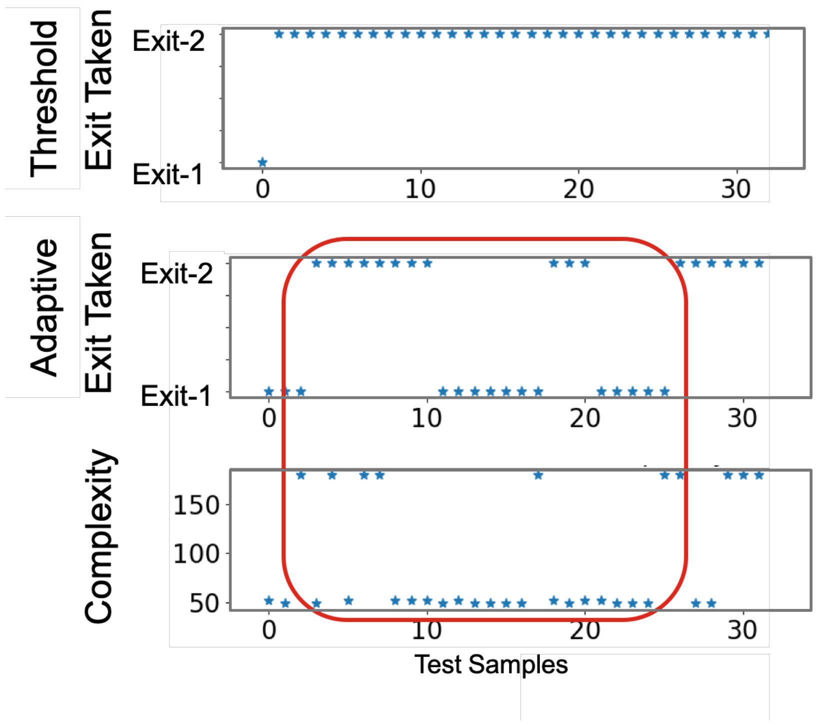


Figure 4.21: Adaptive exit selection with variations in sequential images in DQN-ExitAlexNet. Two different images of same class label with different complexity are modified as sequential frames.

From Fig. 4.21, we see that with threshold-based exit selection, multi-exit DNN uses Exit-2 for the majority of samples and selects Exit-1 for a very few of them based on the threshold checking. As the exit selection is based on checking the threshold at each exit, the temporal relation between the frames is not considered as a part of threshold-based exit strategy. This behavior is seen as the adaptive exit strategy does not switch in immediate response to change in frames but switching happens if change in the exit can provide high accuracy along with computation efficiency. With the adaptive approach, the exit switching happens if there is a scene change in our synthetic video and multiple frames of a new scene are seen by the DQN-Exit. The exit selection is not solely based on the immediate past taken exit but rather a combination of state history and complexity. We see from the Fig. 4.21, for the first ten samples, The DQN prefers exit-2 as more frames of higher complexity are observed. Between samples 10 and 20, lower complexity samples are more frequent, resulting in switching towards Exit-1. Finally, after sample 20, exit switches to Exit-2 as the scene with higher complexity becomes more common.

The analysis shows the importance of complexity as an input state provided to the DQN-Exit network during the training, resulting in an exit policy capable of learning the correlation among sequential inputs. Complexity provides an approximation if two sequential images are similar in difficulty, which is then captured in the history of state vector used by the DQN. This is useful and critical for applications such as agricultural monitoring and building inspection using UAVs [19, 21]. As the scene information does not change randomly the multi-exit DNNs will utilize the same exits for multiple input frames resulting in high computation efficiency.

We utilize the two recent adaptive approaches presented in DynExit [106] and SkipNet [55] to compare performance of our proposed DQN based adaptive exit strategy. DynExit presented the adaptive exit approach for ResNet based DNNs where, they shows 46.3% reduction computation comapred to the end-to-end ResNet architecture. Further, in [106] only single exit branch is used in all of their multi-exit DNNs. In SkipNet [55] a computation cost reduction of 50% is obtained. We observe that our adaptive approach outperforms previous adaptive approaches. Further, adaptive approaches presented in [55, 106] can only be applied to ResNet-based architectures, while our approach applies to different classes of convolutional neural networks, including ResNet and MobileNet, hence, making our approach more widely applicable for many applications and DNNs.

4.8 Distributed Multi-Exit DNN in Edge and Mobile Devices

Here, we consider a scenario where the mobile agents used in an application are low-compute devices with limited processing and storage. These agents may not be capable of storing parameter-heavy multi-exit DNNs. For such cases, network splitting approaches are utilized, where part of the network is stored in the mobile

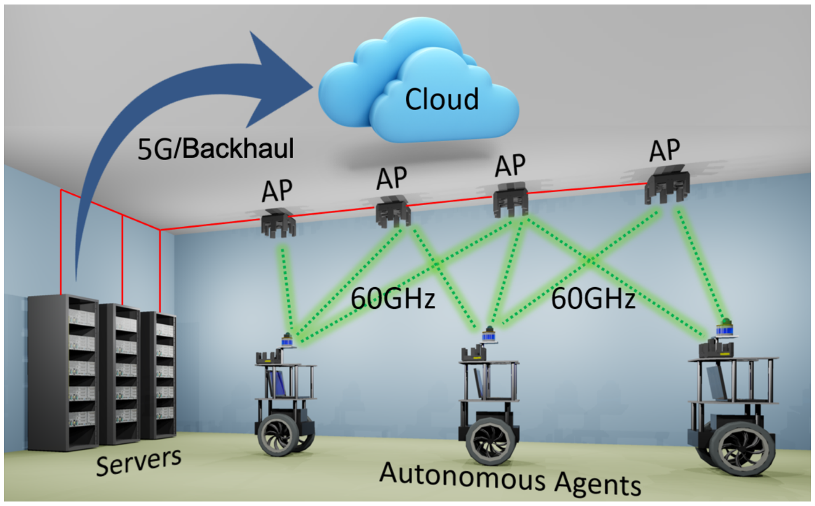
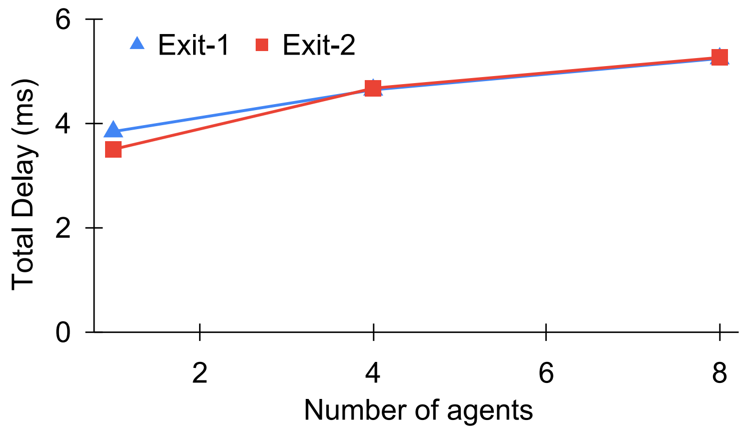


Figure 4.22: 60 GHz based communication between the mobile agents and the local edge server.

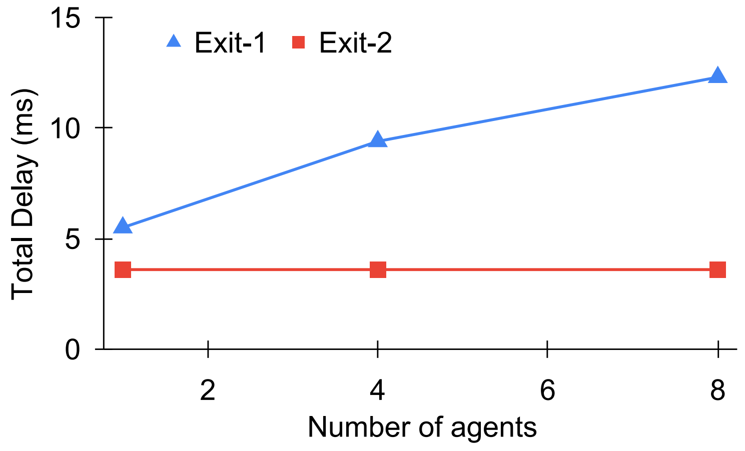
device, and the remaining is stored in a local edge server.

With multi-exit DNNs, the network splitting can be done across the exits. In this way, part of the network up to a specific exit is stored in the mobile agent, and the rest of the network is in the edge server. For inputs that require later exits that are not part of exits stored in the mobile agent, we will need the edge server to perform the execution. During this process, the last layer parameters are also sent to the edge server as the network stored in the edge server will need the previous layer parameter value to continue the execution. With network splitting, the additional cost incurred is the data communication required to send the network parameters associated with the part of the network stored in the mobile device to the edge server to perform the output prediction. The communication infrastructure governs this delay between the devices.

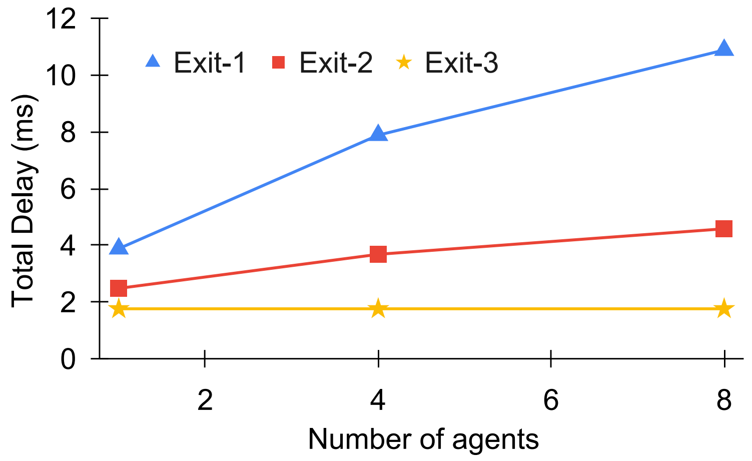
In our analysis, we utilize the high-speed millimeter-wave wireless technology that provides multi-giga-bit data communication between the connected devices. For this, we have considered a scenario where small form factor-based autonomous agents are used. These agents are limited in on-board computation and we utilize multi-exit DNNs for the object classification tasks. The agent store only part of the multi-exit



(a) DQN-ExitAlexNet total inference delay



(b) DQN-ExitMobileNet total inference delay



(c) DQN-ExitResNet total inference delay

Figure 4.23: Total inference delay in a distributed mobile and edge environment

DNN consisting of earlier exits and the rest of the network on a local edge server present inside the warehouse. Figure 4.22 depicts the communication environment that exists between the agents and the server. The autonomous agents are connected to the local edge server using 60 GHz wireless links. A remote cloud can also be used in such architectures where the local edge server communicates with the cloud using the 5G or backhaul connection. The cloud consists of multiple clusters of servers and will offer the most computation resources. But they are remotely located, which results in the highest communication cost. The local edge servers consist of high-performing computation units and are locally connected with the mobile agents using wireless links.

In our evaluation, we only consider the robotic agents and a local edge server in a distributed system along with TP-Link 60 GHz routers as APs. The routers are the same 60 GHz hardware that we used in the design of the millimeter-wave indoor warehouse localization in the first part of the thesis. Authors in [107] performed extensive analysis to study the delay and throughput characteristics using TP-Link 60 GHz routers with varying packet sizes ranging from 1MB to 32 MB. We use the round trip time (RTT) delay associated with the Talon AD7200 routers to evaluate the total delay associated when multi-exit DNNs are used in a distributed environment. We use three scenarios with one, four, and eight agents in the environment and one local edge server. The multi-exit DNNs use the proposed DQN-based exit selection, and the network splitting is done along the exits. We also change the splitting points along all the intermediate exits present in the three multi-exit DNNs, DQN-ExitAlexNet, DQN-ExitMobileNet, and DQN-ExitResNet, to study the effect of the inference delay with the splitting point.

Figure 4.23a, 4.23b, and 4.23c shows the total delay for DQN-ExitAlexNet, DQN-ExitMobileNet, and DQN-ExitResNet with variation in number of mobile agents and network splitting point. For DQN-ExitAlexNet, with splitting at Exit-1, we perform offloading for samples taking Exit-2 and Exit-3. As the number of agents increases, the data transfer latency from agent to edge server increases, resulting in higher inference time. Further, when the splitting is done at Exit-2, only samples taking the Exit-3 need to be offloaded, and samples taking Exit-1 and Exit-2 get executed on the mobile agent.

With DQN-ExitMobileNet, splitting at Exit-1 results in similar observations to DQN-ExitAlexNet. But all the samples either take Exit-1 or Exit-2. No offloading to the edge is required when Exit-2 is selected as the splitting point. Similarly, with DQN-ExitResNet, no offloading is required when splitting is done at Exit-3, as no samples take the last Exit-4. For splitting done at Exit-1 and Exit-2, with increasing offloading, we see an increase in the total inference delay, and the delay is higher with more agents.

4.9 Conclusion

Multi-exit DNNs introduce earlier exits in DNN for samples to exit early if a threshold criterion is met. As the network does not learn the exit selection, it results in sub-optimal exit selection. We proposed a DQN-based adaptive exit selection in multi-exit DNNs. The DQN utilizes various network state information to learn the exit policy, including accuracy, inference latency, previous exit, and image complexity. We implement the DQN-based exit selection on three multi-exit DNNs, Exit-AlexNet, Exit-MobileNet, and Exit-ResNet, representing different network complexities of DNN architectures. Our adaptive exit strategy achieves higher classification accuracy for all three multi-exit DNNs compared to threshold-based exit selection. The adaptive exit selection results in faster inference time, achieving a 14.2% reduction compared to threshold-based exit selection. We also achieve a maximum of 20.1% reduction in total computation cost compared to threshold-based exit selection. Further, compared to traditional end-to-end DNN architectures, the DQN-based exit selection achieves a maximum of 82% reduction in computation cost and a 70% reduction in inference latency. We also present the analysis of network splitting in a distributive edge computing environment using 60 GHz wireless as communication infrastructure. Our approach utilizing DQN-based adaptive exit selection can provide the energy-efficient DNN intelligence required for time-sensitive and computationally constrained devices based on the detailed evaluated performance.

4.10 Chapter Summary

Edge devices have potential to provide autonomy for various mission critical tasks requiring low latency decision making. Such devices lacks the required computational power to run deeper neural networks within the desired latency constraint. Our approach targets at minimizing the computation latency of such DNNs for the tasks of classification by implementing an adaptive exit selection strategy in multi-exit DNNs. In our approach, the multi-exit DNN uses the experience based learning from a DQN agent. This significantly reduces the timing overhead of decision making compared to traditional state-of-the-art threshold-based threshold-based exit strategies. In our approach, the DQN agent learns an intelligent exit policy considering different states that results in reduction in energy consumption, inference time and still maintaining the desired high classification accuracy. Through extensive experimental analysis we have shown the DQN based multi-exit DNNs can achieves maximum of 14.2% reduction in inference time and 20% reduction in computations while achieving higher classification accuracy compared to threshold-based approaches.

Chapter 5

Future Work

Based on the work presented in this dissertation, this chapter discusses possible future directions.

5.1 Hardware and Infrastructure based Information in Adaptive Exit Selection

Adaptive exit selection can utilize more input state information from the hardware and the communication infrastructure to provide more intelligent offloading and exit decisions. DQN can utilize input information to learn exit policy using hardware states such as battery state of charge, computation load, and wireless signal strength. Mobile devices like drones are energy-constrained devices, and an increase in computation resources utilized by running DNNs can negatively affect the battery state of the charge and flight time. Further, in a distributed environment, the offloading decision can be learned based on the quality of the wireless signal strength to perform more adaptive offloading to servers. Figure 5.1 represents the states and approach for a more intelligent exit selection in multi-exit DNNs utilizing hardware and environment state information. Authors in [108] presented a DQN based intelligent offloading of resources from multiple mobile nodes to an edge server. They present a joint offloading decision along that minimizes the energy, delay and the computation cost for offloading. Similar approaches can be investigated in multi-exit DNN based

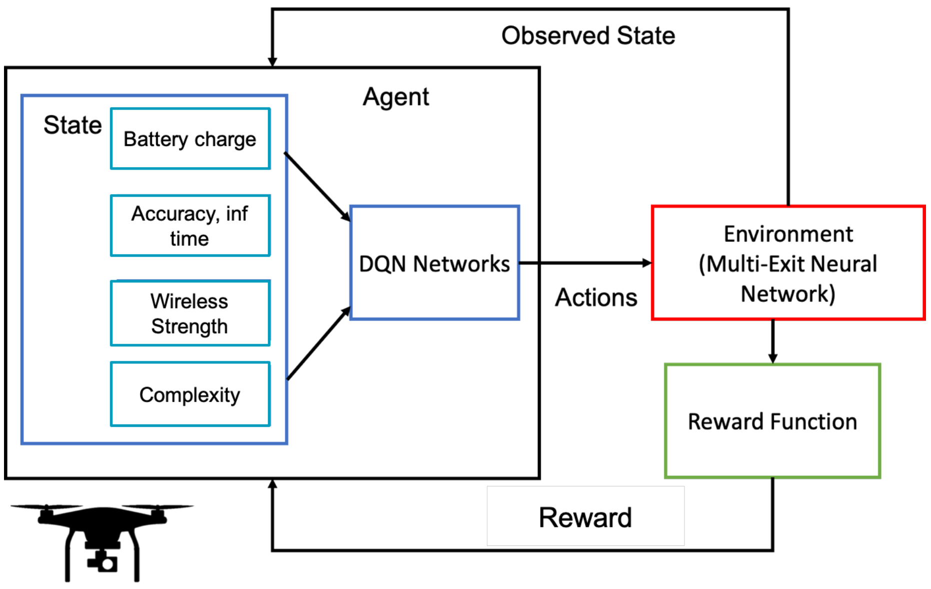


Figure 5.1: Exit selection utilizing more state information

distributed computing environment.

In our initial experiments, we used a simulated battery model to estimate the battery state-of-the-charge (SOC). Modeling the battery-SOC depends on the type of battery used, like lithium-ion and lead-acid. Recent works [109, 110] have introduced battery-SOC models that accurately estimate the battery SOC at future instances. Modeling the battery SOC requires actual measurement values from the battery under various operating conditions and considering most of the physical and chemical processes that occur within the battery. Our work considers the battery model presented in [110] where the battery decay follows a non-linear discharge rate. The modeling may not represent the exact and most accurate representation. We used the reward described in (4.6) and added the battery-SOC as part of the reward.

$$B_{SOC} = 1 - [e^{-cost_{exit}}] \tag{5.1}$$

The decay in the simulated battery-SOC is measured in terms of MAC operation corresponding to the exit taken by the multi-exit DNN represented by $cost_{exit}$ as shown in (5.1). We observe that after including battery-SOC in the reward, the

Table 5.1: Including battery-SOC as an input state

Network	Accuracy (%)	Inf. Time CPU (ms)	Exit (%)
DQN-ExitMobileNet (with battery-SOC)	87	3.1	[20, 80, 0]
DQN-ExitMobileNet	90.78	3.6	[3, 97, 0]

adaptive exit selection increases sample distribution in the earlier exits. This results in a reduction of inference time but also decreases the accuracy achieved by the network. Table 5.1 shows the increase in samples taking the first exit when the battery-SOC is included.

5.2 Federated Learning in Distributed Computing Environment

The idea of federated learning [111] is to provide collaborative learning between distributed devices without sharing the data used to learn the trained ML models. The approach addresses the concerns over privacy and security, as no personal data is being shared for collaborative learning. Only the final trained encrypted model is shared between the nodes and the server. FL is an iterative learning process; the individual ML models at the edge nodes are shared with a central system during each learning episode. The central server uses the local ML models from these individual nodes and performs a unified model update. The model update utilizes the different learning experiences from individual devices performing the same task and shares the updated model with each contributing node. While doing the same ML tasks, each node may see differently distributed data, non-IID, compared to other nodes. This skewed and biased learning data is where FL systems show the enhanced performance improvements.

In FL, there are different possible sources where an attack can be launched. Such sources include edge nodes participating in the FL update process, the central server

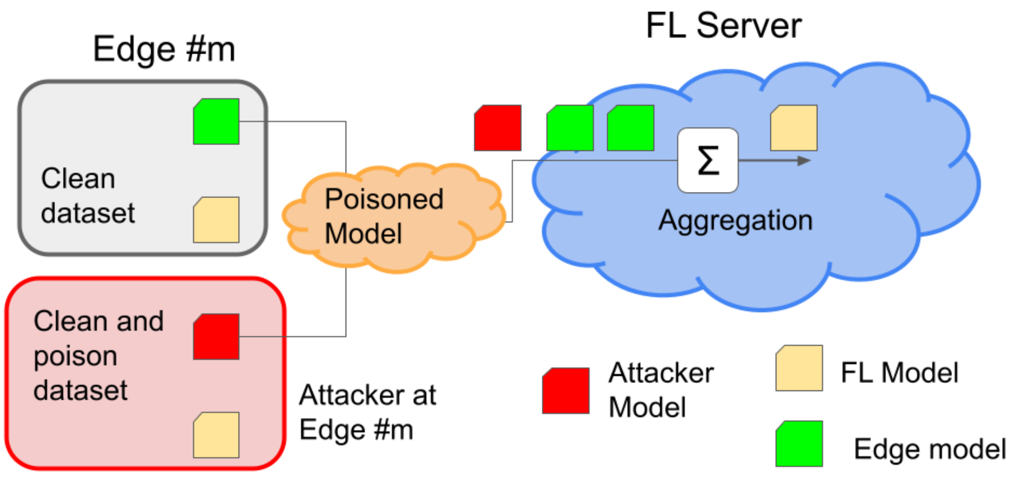


Figure 5.2: Federated learning in distributed environment

performing the model aggregation, and the communication link connecting the edge nodes and the central server. Figure 5.2 illustrates the FL environment where the edge node taking part in the FL update gets affected by the attacker's poisoned data. In such a scenario, the local model sent by the node to the server gets compromised. As a result, the FL server aggregates a poisoned model and sends that as an update to all edge nodes participating in the FL update cycle.

Authors in [112] presented a model poisoning approach showing how a backdoor attack can be launched in FL. They present an attack model where the attacker has access to the local training data at the edge device and creates a malicious model. The work showed such an attack is not detected at the central server and can easily compromise the system. Exploring the usage of federated learning in a distributed environment is a great use case for smart infrastructure. Further, studying different attack models and detection mechanisms in such a scenario is another dimension to extend the current work.

Bibliography

[1] Intel, 2022. [Online]. Available: https://intellabs.github.io/distiller/algo_earlyexit.html

[2] S. Teerapittayanon, B. McDanel, and H. T. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," *CoRR*, vol. abs/1709.01686, 2017. [Online]. Available: http://arxiv.org/abs/1709.01686

[3] W. Zhou, C. Xu, T. Ge, J. McAuley, K. Xu, and F. Wei, "Bert loses patience: Fast and robust inference with early exit," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 18 330–18 341.

[4] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-10 (canadian institute for advanced research)." [Online]. Available: http://www.cs.toronto.edu/~kriz/cifar.html

[5] A. Krizhevsky, "Learning multiple layers of features from tiny images." [Online]. Available: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

[6] B. N. Silva, M. Khan, and K. Han, "Towards sustainable smart cities: A review of trends, architectures, components, and open challenges in smart cities," *Sustainable Cities and Society*, vol. 38, pp. 697–713, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2210670717311125

[7] L. Li, K. Ota, and M. Dong, "Deep learning for smart industry: Efficient manufacture inspection system with fog computing," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 10, pp. 4665–4673, 2018.

[8] D. Brougham and J. Haar, "Smart technology, artificial intelligence, robotics, and algorithms (stara): Employees' perceptions of our future workplace," *Journal of Management amp; Organization*, vol. 24, no. 2, p. 239–257, 2018.

[9] E. Mazareanu, "Total number of warehouses in the u.s. 2007-2020," 2021. [Online]. Available: https://www.statista.com/statistics/873492/total-number-of-warehouses-united-states/

[10] C. Paulausky, "y. death by forklift is really the pits," 2013. [Online]. Available: https://ohsonline.com/Articles/2013/09/01/Deathby-Forklift-is-Really-the-PITs.aspx.

[11] S. Phuyal, D. Bista, and R. Bista, "Challenges, opportunities and future directions of smart manufacturing: A state of art review," *Sustainable Futures*, vol. 2, p. 100023, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666188820300162

[12] K. Shafique, B. A. Khawaja, F. Sabir, S. Qazi, and M. Mustaqim, “Internet of things (iot) for next-generation smart systems: A review of current challenges, future trends and prospects for emerging 5g-iot scenarios,” *IEEE Access*, vol. 8, pp. 23 022–23 040, 2020.

[13] N. Jazdi, “Cyber physical systems in the context of industry 4.0,” in *2014 IEEE International Conference on Automation, Quality and Testing, Robotics*, 2014, pp. 1–4.

[14] J. Shi, J. Wan, H. Yan, and H. Suo, “A survey of cyber-physical systems,” in *2011 International Conference on Wireless Communications and Signal Processing (WCSP)*, 2011, pp. 1–6.

[15] F. Tao, Q. Qi, A. Liu, and A. Kusiak, “Data-driven smart manufacturing,” *Journal of Manufacturing Systems*, vol. 48, pp. 157–169, 2018, special Issue on Smart Manufacturing. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0278612518300062

[16] Amazon, 2022. [Online]. Available: https://www.aboutamazon.com/news/operations/

[17] R. Corporation, 2022. [Online]. Available: https://www.raymondcorp.com/forklifts/automated-lift-trucks

[18] N. Mohamed, J. Al-Jaroodi, I. Jawhar, A. Idries, and F. Mohammed, “Unmanned aerial vehicles applications in future smart cities,” *Technological Forecasting and Social Change*, vol. 153, p. 119293, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0040162517314968

[19] M. Maimaitijiang, V. Sagan, P. Sidike, A. M. Daloye, H. Erkbol, and F. B. Fritschi, “Crop monitoring using satellite/uav data fusion and machine learning,” *Remote Sensing*, vol. 12, no. 9, 2020. [Online]. Available: https://www.mdpi.com/2072-4292/12/9/1357

[20] Z. Cook, L. Zhao, J. Lee, and W. Yim, “Unmanned aerial system for first responders,” in *2015 12th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, 2015, pp. 306–310.

[21] J. Nikolic, M. Burri, J. Rehder, S. Leutenegger, C. Huerzeler, and R. Siegwart, “A uav system for inspection of industrial facilities,” in *2013 IEEE Aerospace Conference*, 2013, pp. 1–8.

[22] J. Levinson, J. Askeland, J. Dolson, and S. Thrun, “Traffic light mapping, localization, and state detection for autonomous vehicles,” in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 5784–5791.

[23] A. Assidiq, O. O. Khalifa, M. R. Islam, and S. Khan, “Real time lane detection for autonomous vehicles,” in *2008 International Conference on Computer and Communication Engineering*, 2008, pp. 82–88.

[24] A. Vashist, D. R. Bhanushali, R. Relyea, C. Hochgraf, A. Ganguly, P. D. Sai Manoj, R. Ptucha, A. Kwasinski, and M. E. Kuhl, "Indoor wireless localization using consumer-grade 60 ghz equipment with machine learning for intelligent material handling," in *2020 IEEE International Conference on Consumer Electronics (ICCE)*, 2020, pp. 1–6.

[25] A. Vashist, M. P. Li, A. Ganguly, S. M. Pd, C. Hochgraf, R. Ptucha, A. Kwasinski, and M. E. Kuhl, "Kf-loc: A kalman filter and machine learning integrated localization system using consumer-grade millimeter-wave hardware," *IEEE Consumer Electronics Magazine*, pp. 1–1, 2021.

[26] M. P. Li, M. E. Kuhl, R. Ballamajalu, C. Hochgraf, R. Ptucha, A. Ganguly, and A. Kwasinski, "Risk-based a: Simulation analysis of a novel task assignment and path planning method," in *2020 Winter Simulation Conference (WSC)*, 2020, pp. 563–571.

[27] N. V. Kumar and C. S. Kumar, "Development of collision free path planning algorithm for warehouse mobile robot," *Procedia Computer Science*, vol. 133, pp. 456–463, 2018, international Conference on Robotics and Smart Manufacturing (RoSMa2018). [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050918310019

[28] L. Li, K. Ota, and M. Dong, "Deep learning for smart industry: Efficient manufacture inspection system with fog computing," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 10, pp. 4665–4673, 2018.

[29] M. Cheffena, "Industrial wireless communications over the millimeter wave spectrum: opportunities and challenges," *IEEE Communications Magazine*, vol. 54, no. 9, pp. 66–72, 2016.

[30] T. S. Rappaport, S. Sun, R. Mayzus, H. Zhao, Y. Azar, K. Wang, G. N. Wong, J. K. Schulz, M. Samimi, and F. Gutierrez, "Millimeter wave mobile communications for 5g cellular: It will work!" *IEEE Access*, vol. 1, pp. 335–349, 2013.

[31] D. Gorecky, M. Schmitt, M. Loskyll, and D. Zühlke, "Human-machine-interaction in the industry 4.0 era," in *IEEE Int. Conf. on Industrial Informatics (INDIN)*, 2014.

[32] A. Diez-Olivan, J. D. Ser, D. Galar, and B. Sierra, "Data fusion and machine learning for industrial prognosis: Trends and perspectives towards industry 4.0," *Information Fusion*, vol. 50, pp. 92–111, 2019.

[33] T. Le-Anh and R. de Koster, "A review of design and control of automated guided vehicle systems," ERIM at Erasmus University Rotterdam, Tech. Rep. ERS;2004-030-LIS, 2004.

[34] Z. Farid, R. Nordin, and M. Ismail, "Recent advances in wireless indoor localization techniques and system," *Journal Comp. Netw. and Communic.*, vol. 2013, pp. 185 138:1–185 138:12, 2013.

[35] X. Guo, L. Li, N. Ansari, and B. Liao, "Accurate wifi localization by fusing a group of fingerprints via a global fusion profile," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 8, pp. 7314–7325, Aug 2018.

[36] N. Yu, X. Zhan, S. Zhao, Y. Wu, and R. Feng, "A precise dead reckoning algorithm based on bluetooth and multiple sensors," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 336–351, Feb 2018.

[37] L. M. Ni, Yunhao Liu, Yiu Cho Lau, and A. P. Patil, "Landmarc: indoor location sensing using active rfid," in *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications, 2003. (PerCom 2003).*, March 2003, pp. 407–415.

[38] K. Sakaguchi, T. Haustein, S. Barbarossa, E. C. Strinati, A. Clemente, G. Destino, A. Pärssinen, I. Kim, H. Chung, J. Kim, W. Keusgen, R. J. Weiler, K. Takinami, E. Ceci, A. Sadri, L. Xain, A. Maltsev, G. K. Tran, H. Ogawa, K. Mahler, and R. W. H. Jr., "Where, when, and how mmwave is used in 5g and beyon," *CoRR*, vol. abs/1704.08131, 2017. [Online]. Available: http://arxiv.org/abs/1704.08131

[39] D. Liang, Z. Zhang, and M. Peng, "Access point reselection and adaptive cluster splitting-based indoor localization in wireless local area networks," *IEEE Internet of Things Journal*, vol. 2, no. 6, pp. 573–585, Dec 2015.

[40] L. Chen, K. Yang, and X. Wang, "Robust cooperative wi-fi fingerprint-based indoor localization," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 1406–1417, Dec 2016.

[41] Y. Liu, Z. Yang, X. Wang, and L. Jian, "Location, localization, and localizability," *Journal of Computer Science and Technology*, vol. 25, no. 2, pp. 274–297, Mar 2010. [Online]. Available: https://doi.org/10.1007/s11390-010-9324-2

[42] Z. Pi and F. Khan, "An introduction to millimeter-wave mobile broadband systems," *IEEE Communications Magazine*, vol. 49, no. 6, pp. 101–107, June 2011.

[43] C. Li, B. Dai, and T. Wu, "Vision-based precision vehicle localization in urban environments," in *2013 Chinese Automation Congress*, Nov 2013, pp. 599–604.

[44] A. Y. Hata and D. F. Wolf, "Feature detection for vehicle localization in urban environments using a multilayer LIDAR," *IEEE Trans. on Intelligent Transportation Systems*, vol. 17, no. 2, pp. 420–429, Feb 2016.

[45] M.-S. Choi and B. Jang, "An accurate fingerprinting based indoor positioning algorithm," 2017.

[46] M. Brunato and R. Battiti, "Statistical learning theory for location fingerprinting in wireless lans," *Computer Networks*, vol. 47, no. 6, pp. 825 – 845, 2005.

[47] Z. Wu, C. Li, J. K. Ng, and K. R. p. h. Leung, "Location estimation via support vector regression," *IEEE Transactions on Mobile Computing*, vol. 6, 2007.

[48] Y. Li, X. Hu, Y. Zhuang, Z. Gao, P. Zhang, and N. El-Sheimy, "Deep reinforcement learning (drl): Another perspective for unsupervised wireless localization," *IEEE Internet of Things Journal*, pp. 1–1, 2019.

[49] T. Koike-Akino, P. Wang, M. Pajovic, H. Sun, and P. V. Orlik, "Fingerprinting-based indoor localization with commercial mmwave wifi: A deep learning approach," *IEEE Access*, Apr. 2020. [Online]. Available: https://www.merl.com/publications/TR2020-054

[50] F. Dellaert, D. Fox, W. Burgard, and S. Thrun, "Monte carlo localization for mobile robots," in *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, vol. 2, 1999, pp. 1322–1328 vol.2.

[51] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[52] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *arXiv preprint arXiv:1512.03385*, 2015.

[53] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017. [Online]. Available: http://arxiv.org/abs/1704.04861

[54] E. Demir, "Early-exit convolutional neural networks," Ph.D. dissertation, Middle East Technical University, 2019. [Online]. Available: https://github.com/eksuas/eenets.pytorch

[55] X. Wang, F. Yu, Z.-Y. Dou, T. Darrell, and J. E. Gonzalez, *SkipNet: Learning Dynamic Routing in Convolutional Networks.* Cham: Springer International Publishing, 2018, pp. 420–436.

[56] S. Jain, S. Venkataramani, V. Srinivasan, J. Choi, P. Chuang, and L. Chang, "Compensated-dnn: Energy efficient low-precision deep neural networks by compensating quantization errors," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, 2018, pp. 1–6.

[57] M. Courbariaux and Y. Bengio, "Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1," *CoRR*, vol. abs/1602.02830, 2016. [Online]. Available: http://arxiv.org/abs/1602.02830

[58] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013. [Online]. Available: http://arxiv.org/abs/1312.5602

[59] M. Cheng, J. Li, and S. Nazarian, "Drl-cloud: Deep reinforcement learning-based resource provisioning and task scheduling for cloud service providers," in *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2018, pp. 129–134.

[60] M. Babaeizadeh, I. Frosio, S. Tyree, J. Clemons, and J. Kautz, "GA3C: gpu-based A3C for deep reinforcement learning," *CoRR*, vol. abs/1611.06256, 2016. [Online]. Available: http://arxiv.org/abs/1611.06256

[61] Y. F. Chen, M. Everett, M. Liu, and J. P. How, "Socially aware motion planning with deep reinforcement learning," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 1343–1350.

[62] S. Krishnan, P. Sharma, Z. Guoping, and O. H. Woon, "A uwb based localization system for indoor robot navigation," in *2007 IEEE International Conference on Ultra-Wideband*, 2007, pp. 77–82.

[63] S. J. Velat, J. Lee, N. Johnson, and C. D. Crane, "Vision based vehicle localization for autonomous navigation," in *2007 International Symposium on Computational Intelligence in Robotics and Automation*, 2007, pp. 528–533.

[64] M. Giordani, A. Zanella, and M. Zorzi, "Millimeter wave communication in vehicular networks: Challenges and opportunities," in *2017 6th International Conference on Modern Circuits and Systems Technologies (MOCAST)*, May 2017, pp. 1–6.

[65] A. M. Ladd, K. E. Bekris, A. Rudys, L. E. Kavraki, and D. S. Wallach, "Robotics-based location sensing using wireless ethernet," *Wireless Networks*, vol. 11, no. 1, pp. 189–204, Jan 2005. [Online]. Available: https://doi.org/10.1007/s11276-004-4755-8

[66] M. Cheffena, "Industrial wireless communications over the millimeter wave spectrum: opportunities and challenges," *IEEE Communications Magazine*, vol. 54, no. 9, pp. 66–72, Sep. 2016.

[67] M. Cheffena, "Industrial wireless sensor networks: channel modeling and performance evaluation," *EURASIP Journal on Wireless Communications and Networking*, vol. 2012, no. 1, p. 297, Sep 2012. [Online]. Available: https://doi.org/10.1186/1687-1499-2012-297

[68] F. Zhao, T. Huang, and D. Wang, "A probabilistic approach for wifi fingerprint localization in severely dynamic indoor environments," *IEEE Access*, vol. 7, pp. 116 348–116 357, 2019.

[69] T. S. Rappaport, Y. Xing, G. R. MacCartney, A. F. Molisch, E. Mellios, and J. Zhang, "Overview of millimeter wave communications for fifth-generation (5g) wireless networks—with a focus on propagation models," *IEEE Transactions on Antennas and Propagation*, vol. 65, no. 12, pp. 6213–6230, Dec 2017.

[70] G. Bielsa, J. Palacios, A. Loch, D. Steinmetzer, P. Casari, and J. Widmer, "Indoor localization using commercial off-the-shelf 60 ghz access points," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, April 2018, pp. 2384–2392.

[71] "Tp-link ad7200," https://www.tp-link.com/us/home-networking/wifi-router/ad7200/, accessed: 2010-09-13.

[72] L. Perez and J. Wang, "The effectiveness of data augmentation in image classification using deep learning," *CoRR*, vol. abs/1712.04621, 2017. [Online]. Available: http://arxiv.org/abs/1712.04621

[73] D. Pauluzzi and N. Beaulieu, "A comparison of snr estimation techniques for the awgn channel," *IEEE Transactions on Communications*, vol. 48, no. 10, pp. 1681–1691, 2000.

[74] M. A. Suliman, A. M. Alrashdi, T. Ballal, and T. Y. Al-Naffouri, "Snr estimation in linear systems with gaussian matrices," *IEEE Signal Processing Letters*, vol. 24, no. 12, pp. 1867–1871, 2017.

[75] B. Yazici and S. Yolacan, "A comparison of various tests of normality," *Journal of Statistical Computation and Simulation*, vol. 77, no. 2, pp. 175–183, 2007. [Online]. Available: https://doi.org/10.1080/10629360600678310

[76] A. Subramanian, P. Deshpande, J. Gao, and S. Das, "Drive-by localization of roadside wifi networks," in *IEEE INFOCOM 2008 - The 27th Conference on Computer Communications*, 2008, pp. 718–725.

[77] M. Z. Comiter, M. B. Crouse, and H. T. Kung, "A data-driven approach to localization for high frequency wireless mobile networks," in *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, 2017, pp. 1–7.

[78] *Anderson–Darling Test*. New York, NY: Springer New York, 2008, pp. 12–14. [Online]. Available: https://doi.org/10.1007/978-0-387-32833-1_11

[79] S. Rezaei and R. Sengupta, "Kalman filter-based integration of dgps and vehicle sensors for localization," *IEEE Transactions on Control Systems Technology*, vol. 15, no. 6, pp. 1080–1088, 2007.

[80] Z. Chen, H. Zou, H. Jiang, Q. Zhu, Y. C. Soh, and L. Xie, "Fusion of wifi, smartphone sensors and landmarks using the kalman filter for indoor localization," *Sensors*, vol. 15, no. 1, pp. 715–732, 2015. [Online]. Available: https://www.mdpi.com/1424-8220/15/1/715

[81] “Simcona electronics corporation,” https://www.simcona.com/, accessed: 2020-02-23.

[82] P. Bahl and V. N. Padmanabhan, “RADAR: an in-building rf-based user location and tracking system,” in *IEEE INFOCOM*, 2000.

[83] C. Laoudias, P. Kemppi, and C. G. Panayiotou, “Localization using radial basis function networks and signal strength fingerprints in wlan,” in *GLOBECOM 2009 - 2009 IEEE Global Telecommunications Conference*, Nov 2009, pp. 1–6.

[84] Z. Yang, C. Wu, and Y. Liu, “Locating in fingerprint space: Wireless indoor localization with little human intervention,” in *International Conference on Mobile Computing and Networking*, 2012.

[85] O. Kanhere and T. S. Rappaport, “Position locationing for millimeter wave systems,” *CoRR*, vol. abs/1808.07094, 2018. [Online]. Available: http://arxiv.org/abs/1808.07094

[86] Z. Wei, Y. Zhao, X. Liu, and Z. Feng, “Doa-lf: A location fingerprint positioning algorithm with millimeter-wave,” *IEEE Access*, vol. 5, pp. 22 678–22 688, 2017.

[87] W. Y. Al-Rashdan and A. Tahat, “A comparative performance evaluation of machine learning algorithms for fingerprinting based localization in dm-mimo wireless systems relying on big data techniques,” *IEEE Access*, vol. 8, pp. 109 522–109 534, 2020.

[88] Z. Li, Z. Tian, Z. Wang, and Z. Zhang, “Multipath-assisted indoor localization using a single receiver,” *IEEE Sensors Journal*, vol. 21, no. 1, pp. 692–705, 2021.

[89] M. Erdelj, M. Król, and E. Natalizio, “Wireless sensor networks and multi-uav systems for natural disaster management,” *Computer Networks*, vol. 124, pp. 72–86, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128617302220

[90] C. Szegedy, S. Ioffe, and V. Vanhoucke, “Inception-v4, inception-resnet and the impact of residual connections on learning,” *CoRR*, vol. abs/1602.07261, 2016. [Online]. Available: http://arxiv.org/abs/1602.07261

[91] J. Redmon and A. Farhadi, “Yolov3: An incremental improvement,” *CoRR*, vol. abs/1804.02767, 2018. [Online]. Available: http://arxiv.org/abs/1804.02767

[92] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, “Learning transferable architectures for scalable image recognition,” *CoRR*, vol. abs/1707.07012, 2017. [Online]. Available: http://arxiv.org/abs/1707.07012

[93] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg, “SSD: single shot multibox detector,” *CoRR*, vol. abs/1512.02325, 2015. [Online]. Available: http://arxiv.org/abs/1512.02325

[94] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," *CoRR*, vol. abs/1506.01497, 2015. [Online]. Available: http://arxiv.org/abs/1506.01497

[95] A. L. Imoize, O. Adedeji, N. Tandiya, and S. Shetty, "6g enabled smart infrastructure for sustainable society: Opportunities, challenges, and research roadmap," *Sensors*, vol. 21, no. 5, 2021. [Online]. Available: https://www.mdpi.com/1424-8220/21/5/1709

[96] J. Wang, Z. Feng, Z. Chen, S. George, M. Bala, P. Pillai, S.-W. Yang, and M. Satyanarayanan, "Bandwidth-efficient live video analytics for drones via edge computing," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, 2018, pp. 159–173.

[97] X. Dai, X. Kong, and T. Guo, "Epnet: Learning to exit with flexible multi-branch network," in *Proceedings of the 29th ACM International Conference on Information Knowledge Management*, ser. CIKM '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 235–244. [Online]. Available: https://doi.org/10.1145/3340531.3411973

[98] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," *CoRR*, vol. abs/1405.3866, 2014. [Online]. Available: http://arxiv.org/abs/1405.3866

[99] S. Srinivas and R. V. Babu, "Data-free parameter pruning for deep neural networks," *CoRR*, vol. abs/1507.06149, 2015. [Online]. Available: http://arxiv.org/abs/1507.06149

[100] Y. Wu, Z. Wang, Z. Jia, Y. Shi, and J. Hu, "Intermittent inference with nonuniformly compressed multi-exit neural network for energy harvesting powered devices," in *Proceedings of the 57th ACM/EDAC/IEEE Design Automation Conference*, ser. DAC '20. IEEE Press, 2020.

[101] W. Jiang, X. Zhang, E. H. Sha, L. Yang, Q. Zhuge, Y. Shi, and J. Hu, "Accuracy vs. efficiency: Achieving both through fpga-implementation aware neural architecture search," *CoRR*, vol. abs/1901.11211, 2019. [Online]. Available: http://arxiv.org/abs/1901.11211

[102] C. Olah, A. Mordvintsev, and L. Schubert, "Feature visualization," *Distill*, 2017, https://distill.pub/2017/feature-visualization.

[103] C.-H. Teh and R. Chin, "On the detection of dominant points on digital curves," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, no. 8, pp. 859–872, 1989.

[104] H. Huiping, W. Bingfang, and F. Jinlong, "Analysis to the relationship of classification accuracy, segmentation scale, image resolution," in *IGARSS 2003. 2003 IEEE International Geoscience and Remote Sensing Symposium. Proceedings (IEEE Cat. No.03CH37477)*, vol. 6, 2003, pp. 3671–3673 vol.6.

[105] S. Hao, Y. Cui, and J. Wang, “Segmentation scale effect analysis in the object-oriented method of high-spatial-resolution image classification,” *Sensors*, vol. 21, no. 23, 2021. [Online]. Available: https://www.mdpi.com/1424-8220/21/23/7935

[106] M. Wang, J. Mo, J. Lin, Z. Wang, and L. Du, “Dynexit: A dynamic early-exit strategy for deep residual networks,” in *2019 IEEE International Workshop on Signal Processing Systems (SiPS)*, 2019, pp. 178–183.

[107] H. Assasa, S. Kumar Saha, A. Loch, D. Koutsonikolas, and J. Widmer, “Medium access and transport protocol aspects in practical 802.11 ad networks,” in *2018 IEEE 19th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*, 2018, pp. 1–11.

[108] L. Huang, X. Feng, C. Zhang, L. Qian, and Y. Wu, “Deep reinforcement learning-based joint task offloading and bandwidth allocation for multi-user mobile edge computing,” *Digital Communications and Networks*, vol. 5, no. 1, pp. 10–17, 2019, artificial Intelligence for Future Wireless Communications and Networking. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352864818301469

[109] B. Homan, G. J. M. Smit, R. P. van Leeuwen, M. V. ten Kortenaar, and B. V. Ten, “A comprehensive model for battery state of charge prediction,” in *2017 IEEE Manchester PowerTech*, 2017, pp. 1–6.

[110] B. Homan, M. V. ten Kortenaar, J. L. Hurink, and G. J. Smit, “A realistic model for battery state of charge prediction in energy management simulation tools,” *Energy*, vol. 171, pp. 205–217, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0360544218325064

[111] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečný, S. Mazzocchi, B. McMahan, T. Van Overveldt, D. Petrou, D. Ramage, and J. Roselander, “Towards federated learning at scale: System design,” in *Proceedings of Machine Learning and Systems*, A. Talwalkar, V. Smith, and M. Zaharia, Eds., vol. 1, 2019, pp. 374–388. [Online]. Available: https://proceedings.mlsys.org/paper/2019/file/bd686fd640be98efaae0091fa301e613-Paper.pdf

[112] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, “How to backdoor federated learning,” *CoRR*, vol. abs/1807.00459, 2018. [Online]. Available: http://arxiv.org/abs/1807.00459