Real-Time Data processing and AI for Distributed IoT

- Project Proposal -

Amila Weerasinghe
Rajitha Opanayaka
Rashmi Thilakarathne

Department of Computer Engineering
University of Peradeniya

Final Year Project (courses CO421 & CO425) report submitted as a requirement of the degree of
B.Sc.Eng. in Computer Engineering

October 2020

Supervisors: Dr. Upul Jayasinghe (University of Peradeniya) and Dr.Damayanthi Herath (University of Peradeniya)

We would like to dedicate this thesis to our loving parents and “teachers” . . .
Who never gave up on us
through thick and thin. . .

Declaration

We hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is our own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgments.

Amila Weerasinghe
Rajitha Opanayaka
Rashmi Thilakarathne
October 2020

Acknowledgements

We would like to express our sincere gratitude to our supervisors Dr.Upul Jayasinghe and Dr.Damayanthi Herath for providing their invaluable guidance, comments and suggestions throughout the course of the project. Especially the patience, motivation, and immense knowledge of Dr.Upul Jayasinghe accompanied by Dr.Damayanthi Herath guided us thorough the journey. We also place on record, our sense of gratitude to one and all, who directly or indirectly, have lent their hand in this venture.

Abstract

Artificial Intelligence has been impacted in a variety of industries, leading the world towards revolutionary applications and services that are primarily driven by high-performance computation and storage facilities in the cloud.This is mainly due to the advantage of having higher computational power, larger storage capacity, and scalability. But with the increase of millions of IoT devices, a huge amount of data is being generated by end devices. To process such data, the distributed end devices have to communicate with the cloud servers making it difficult to generate real-time decisions though it consumes a lot of resources including bandwidth, processing, and storage facilities at the cloud. On the other hand, edge computing architectures which consist of Fog computing and ROOF computing enable a distributed way to process data near the sources of data which leads to facilitate real-time processing. But with the limited resources in the end devices, it is quite challenging to perform complex AI algorithms. Thus by developing AI algorithms that can perform AI processing with limited resources with the advantage of Fog computing and ROOF computing, distributed systems like IoT can harvest to its true potentials including real-time decision making and smart systems that lead to new business models while saving computational resources, bandwidth, and storage facilities at the cloud.

Table of contents

List of figures

List of tables

Nomenclature

Acronyms / Abbreviations

AI	Artificial Intelligence
CNN	Convolutional Neural Network
CPE	Cognitive Processing Elements
CPU	Central Processing Unit
DRL	Deep Reinforcement Learning
GPU	Graphical Processing Unit
IoT	Internet of Things
IOU	Intersection over union
ML	Machine Learning
RAM	Random Access Memory
ROOF	Real-time Onsite Operations Facilitation
YOLO	You Only Look Once

Chapter 1

Introduction

Today IoT has a huge impact upon the large scope of fields. Millions and billions of data generated with the development of the Internet of things [3]. Also the development of Artificial Intelligence has extended the technology capabilities. To achieve the maximum capabilities, IoT is enriched with Artificial intelligence. But most of the existing Artificial algorithms require more computational power, Storage, bandwidth and resources. When it comes to distributed architecture IoT, basically cloud computing provides the infrastructure which fulfills the needs of distributed IoT. In some scenarios real time decision taking from the end devices is a must.

Cloud architecture provides infrastructure at a higher level. Therefore to process data in cloud computing level, data has to be transmitted from end devices to the core cloud servers [4]. Network congestion, latency in the communication causes the data processing and decision taking in real time impossible. So a new paradigm has to be introduced in order to overcome the real time restrictions in Artificial intelligence based distributed IoT. Edge computing allows computation to be done in a closer proximity to end devices where only limited resources are available. They can offload the computation when a certain threshold value exceeds, Where policy technology can be used to make decisions associated with offloading. DRL can be used jointly in making decisions based upon Deep Q-networks. Where Federated Learning allows to train the DRL agents. By using both Federated learning and Collaborative Federated learning techniques computation can be done using local data, Where combination of these techniques leads to real time processing in distributed IoT.

Edge computing allows computation to be done in a closer proximity to end devices where only limited resources are available. They can offload the computation when a certain threshold value exceeds Where policy technology can be used to make decisions associated with offloading. DRL can be used jointly in making decisions based upon

Deep Q-networks, Where Federated Learning allows to train the DRL agents. By using both Federated learning and Collaborative Federated learning techniques computation can be done using local data Where combination of these techniques leads to real time processing in distributed IoT.

1.1 Background

Accomplishing the goal of handling Real time AI and ML processing at the edge devices in distributed IoT is based on various aspects. The taxonomy of Cloud computing, Fog Computing and ROOF Computing is a major concept. When it comes to the background the learning techniques and parameter sharing plays a major role. It is vital to adhere to the limited resources in edge devices and application of policy technology in distributed IoT.

1.1.1 Edge Computing

In 2012 to address the challenges of IoT applications the concept of Fog computing was first introduced by Cisco. Compared to the cloud servers, ROOF [5] and fog [6][7] is much closer to the user. Fog computing enables which are the main features of cloud computing, IaaS(Infrastructure as a Service), PaaS(Platform as a Service) and SaaS(Software as a service), to extend towards the edge devices in more close proximity.

Fog nodes can be described under three types namely:

- Servers which are are geo-distributed and are deployed at very common places
- Networking Devices like gateway routers, switches and set-top boxes
- Cloudlets which are considered as micro clouds

Also there should be a collaboration between these nodes. Collaboration techniques can be categorised into three groups.They are cluster, peer to peer and master-slave. In clustering, Clusters can be formed by the homogeneity of fog nodes [8]. Clusters can be static or dynamic. In static clusters they are difficult to scalable in runtime and dynamic formation of clusters have to considered in terms of network overhead.Peer to peer collaboration can be in both hierarchical and flat order based on nearness. In Master-Slave approach basically master Fog node have the main control i.e it controls functionalities, processing load, resource management, data flow, etc. of underlying slave nodes However, in real-time data processing due to this kind master and the slave Fog nodes require high bandwidth to communicate with each other. Even though resources are limited compared to the cloud by reducing the transmission delays and using resources efficiently real time processing can be achieved. Offloading and caching techniques can be used to increase the real time processing capabilities at the edge.

1.1.2 Deep Reinforcement Learning

Deep Reinforcement Learning(DRL) [9][10] is used to jointly manage the communication and computation resources. Capabilities of DRL can even be used to train Deep Q-Learning algorithms [11] and use them as the machine learning model at the edge [3]. Also Federated Learning can be used as a framework for training DRL agents. In this technique, DRL can be used with basically two scenarios. They are with caching [12] and with offloading. In caching DRL, popular content is cached in the intermediate servers or routers. Caching is done so that the most accessed or popular content can be accessed easily. To determine what is to be cached exactly the approach is to define the popularity of files. Basically, the popularity is defined by the probability distribution of the number of requests from users. Caching can be modeled as a Markov Decision Process and then use Deep Reinforcement Learning to Solve it is a compatible approach. In offloading DRL the communication is modelled as a finite state discrete Markov chain [4]. Then computational tasks are formed as a bernoulli distribution. Those tasks are queued as served in FIFO(First In First Out) structure. The problem formulation is done and handled via a stationary control policy.

1.1.3 Policy technology

In most of the works the distributed IoT is served by policy technology in various aspects. The reason for having the policy technology is that the end devices have different capabilities. In top down approaches, which the AI model is trained in the cloud using high resources and then the AI algorithm is deployed to the edge. Few variations of AI models are trained as a general. Out of these few the most suitable AI model should be deployed into the suitable device depending on the different capabilities. For the purpose at the edge the policy technology is used to apply the most appropriate model. In edge the AI algorithms have to be updated depending on the input reading from the edge devices. Depending on different environments the model might be updated in various ways.There should be a system that generalises the distributed models, So that the updated models can be used to merge together in a suitable format. That systems can be implemented using the policy technologies. The policies can be implemented on edge devices so that the different models can be learnt on the edge and offloaded to the fog computing level if more resources are needed for computing.

1.1.4 Collaborative Federated Learning

Due to the resource limitations of the devices, devices may not be able to connect to the base station. Global model may be not highly precise. Overcome these problems Collaborative FL [13] can be used where if the device cannot connect to the base station it can connect to the nearest device and train the model. In this scenario each device must be able to aggregate the each local model of other devices and train its own local model. CFL performance may vary with network topology from with the nearest devices.Centralized and distributed solutions can be used to form network topology. Distributed solutions are more practical which can be implemented using network formation games.

1.1.5 Federated learning

Due to the data privacy and resource constraints centralize learning may not be able to give the best performance. Overcome these constraints federated learning [14][15][12] [16] techniques can be used where each device trains its own local model without sharing its data. Only the model parameters are sent to the Base Station to train a global model. Algorithms to enable Federated learning, FedAvg [17] runs SGD in user devices(IoT) and average local weights in a server to train a global model while FedProx is an improved version of FedAvg but both algorithms weighted all devices equally. q-FedAvg algorithm gives more weights to the limited resources devices while aggregating the model parameter from the end devices.

1.2 The problem

When Artificial Intelligence models are used in distributed IoT, edge devices have only constrained limited capabilities. Edges IoT devices have limited storage, Computational power. As Cloud computers supply IaaS (Infrastructure As A Service) clouds are rich with high computational resources. Therefore the traditional way of doing AI and ML computation is cloud base In IoT the sensors and actuators are connected to edge devices [3]. Therefore to acquire rich computational resources data has to be communicated over networks to the cloud [18]. But in the communication from edge to cloud the network congestion causes communication latency. Because millions of end devices which generate huge amounts of data need to communicate with cloud servers consuming lots of bandwidth that generate network congestion. Therefore the Real time AI is not performed on the edges of this architecture. When the computation is moved towards the edge. It reduces the network congestions. But in those levels resources are limited. With limited resources such as computation power and storage it makes difficult to generate real time decisions in AI or ML algorithms which requires high computational resources.

1.2.1 The proposed solution

We introduce a methodology for Real time processing at the edge for AI [19] algorithms harvesting the advantages of distributed architecture. Instead of using high computational resources at the cloud, processing is to be done at the edges with limited available resources. Basically the AI model will be processed at the edge, if the available resources are not enough for processing the computation will be distributed to the cluster of available edge nodes. The distribution of the computation is based upon the map reduce like architecture. If the edge nodes are incapable of processing the assigned computation, that particular task will be offloaded [20][21] to the ROOF computing level and will be distributed to the other nodes of the cluster. The offloading will be based upon policy technology that is to be defined. Use of the ROOF computing and fog computing [8] avoids the necessity of reaching the cloud servers. Therefore the processing can be done with lesser latency. That particular architecture leads to achieve goal for real time processing at the edge. When nodes in the cluster processes the federated learning approach is used to process the nodes separately as well as to use ROOF computing level to train global models without accessing the cloud computing level. The whole methodology can be used to accomplish the goal of handling AI and ML computation with limited resources while harvesting the advantages of a distributed architecture for real-time processing at the edge [3].

Chapter 2

Related work

Handling AI and ML computation with limited resources while harvesting the advantages of a distributed architecture for real-time processing at the edge has a scope which is associated with various contexts.

2.1 Use of Map Reduce to distribute the computation

The distribution of computation can be categorised basically under three main scenarios [22]:

- When the training data is large.
- When data to be classified is large.
- When the Neural network consists of a huge number of nodes.

When the training data set is larger training data has to be divided using mapper function is Map reduce [23]. Because training a huge volume of data costs high computational resources as well as a high amount of time. So different sets of data will be provided to each node in the cluster and the whole portion of data to be classified will be provided to every node, because the volume of data to be classified is small compared to the volume of training data. When each node in the cluster is trained with different chunks of training data. That will generate different AI models at the different nodes of the cluster. So federated learning techniques [16] should be used in order to define the suitable training model out of the different models that will be generated.

When data to be classified is huge that particular data will be distributed to all the nodes using the mapper function. For each node in the cluster the same chunk of training data will be provided because the training data is small compared to data to be classified in this scenario. Each node will generate its output and the reducer function will make the final output from all of these results.

When the neural network consists of a large number of neurons, computation cost increases. So using map reduce the neural network can be distributed over multiple nodes. There are a number of iterations involved with the feedforward process while the backpropagation is done in the last iteration. In each iteration reducer collects the outputs and merges all outputs from each mapper. For this approach computers with high computation power are used. In this work clusters consist of low computation power nodes.

2.2 Top Down based approach

Basically in top down approach the AI model is trained or developed at the cloud computing level using the high computational resources.Then the model is deployed to the edge devices.There are various techniques used in the scenarios. The whole process can be considered as a tree like structure. Root is considered as the central cloud and the leaves of the tree are the edge devices. A technique to look at the AI application built is that it is considered as the Cognitive Processing Elements(CPE). Then build a chain of CPE. A CPE is operated in basic four phases [24]:

- Discover phase
- Deploy phase
- Operate phase
- Retain phase

Basically in the discovery phase the basic idea is to develop the model using automated or user defined technique. In some scenarios multiple models might be generated and there should be methods to decide and select the best models out of those. After the model is developed they should be added into edges using the Deploy phase. Here the model is packed into docker containers and placed in a shared repository. Therefore edges can access that particular repository. Above mentioned chain of CPE is implemented as a Node-Red flow.

Under the operation phase microservices are implemented on edges which are responsible for instantiating the chain of CPE flow. Docker containers in the shared repository will be executed at the edges by their microservices. Retaining phase is designed such that feedback mechanisms will be provided towards the cloud and then models can be updated in the cloud computing level.

This top down approach uses high computational resources available at the cloud level and then deploys the models to edges. In contrast what we propose is to use the limited resources available at the edges, so our solution is a bottom up approach.

2.3 Edge and Fog Computing Enabled AI for IoT

By defining the edge and the fog nodes which are intermediate nodes between the edge and the cloud, these fog nodes can be used to increase the efficiency of distributed AI algorithms. This approach is based on containerization. Due to containerization, the containers can be migrated horizontally and vertically between the cloud computing layer and the edge. This architecture leads to more flexibility of Fog computing layer [25]. This method using containers it can execute and classify locally in nodes and offload to the fog and the cloud whenever additional computation is required. The approach uses the Distributed Deep Neural Networks. We can use the ideology to use fog computing. But to enable real time computations, use of both ROOF computing and Fog Computing [8] can be much efficient. This approach is focused on hardware architectural solutions in the terms of making energy efficiency rather than developing algorithmic solutions which we focused on achieving.

2.4 Fog Computing

Defining the fog nodes in Fog Computing and the methods to Collaborate the Fog Nodes [8] are described;

- Cluster
- Peer to Peer
- Master Slave

Cluster architecture describes the way to form a collaborative execution environment by forming a cluster of nodes.Peer to Peer architecture can be formed in hierarchical manner as well as in a flat order. Master slave designed such a way that master fog controls the process load, resource management of slave nodes. The collaboration methods of the Fog Computing can be used on our node collaboration in ROOF Computing and Fog Computing. Limitations of the architecture is that master-slave uses high bandwidth between master and slaves. Therefore the better approach would be to define a hybrid methodology consisting of master-slave along with peer to peer and cluster.

2.5 Caching and Communication by Federated Learning approach

Basically tries to integrate Deep Reinforcement Learning Techniques with Federated Learning Framework. The idea of exchanging the learning parameters for better training and inferencing the model is described. The approach has used the techniques to reduce the system's communication load. Deep Reinforcement Learning(DRL) is used to jointly manage communication and the computational resources. While Federated Learning is used as a Framework for training the DRL agents [4]. The distributed architecture leads to different updates to model at different nodes, because of the independent training. So the necessity of merging the updates arises. There are problems associated with the models in the nodes as the nodes are not perfectly similar. Therefore this unbalance of devices has to be figured out, As a solution this research proposes to use FedAvg algorithm [17]. We can use this algorithm to aggregate the global model at a ROOF or the fog level based on the different models developed at the edges.

Chapter 3

Methodology

Our main goal is here to process real time data in using Distributed IoT. But the main issues in IoT devices are they have very low processing power and memory. So it's very hard to run AL algorithms in IoT devices due to processing and memory bandwidth issues. So the current approach is with the help of cloud computing [26] we can distribute the processing workload to the cloud [18] and get back the result in the edge layer, using this method we can reduce the processing time it takes to compute the algorithm. But the main downside of this approach is the time it takes to upload and data to the cloud and download data from the cloud. So this time delay may affect the real time data processing of the edge.

To overcome this problem, a fog layer is introduced [27], a new layer is added in between the cloud and the edge layers. So instead of sending data to the cloud and processing it in the cloud, now data is sent to the fog layer, this will reduce the upload and download time reducing the time it takes to send and receive data, this helps to archive near real-time data processing for IoT devices. But still the issue of real time processing is not archived. The main issue is this technique is the edge device is always offloading the data to the cloud which causes a delay.

what if we can set parameters to define when to offload the data to cloud and try to process the data in real-time if possible. This is known as policy technology [28]. By utilizing this method, the edge device can decide when to offload the data according to their capability of the IoT device. We can set different threshold parameters to when to decide the offloading. Usual threshold parameter is 80% of total processing power. When processing power reaches to 80% then the device will offload the work to the cloud, unless it will process in real time. So this method will only offload when it's necessary so we can archive near real-time results.

Few factors will come into play when it come to deciding the threshold parameter, if we use a lower threshold parameter then, then it will more often try to reach that value so the edge device will more likely to offload the data to upper layers resulting the more often data offloading which will affect the real time data processing of the edge device. And there is a resource wastage in the edge as well. Because it can go for higher processing without any bottleneck of the processing capabilities. And if we go for a very high threshold parameter that can affect the result as well. Because when the algorithm is running on the edge and it's reached the threshold parameter limit, we may run out of time to send data to the upper layers. so it causes a time delay in processing which affects the real time process. We need to find the right balance between the threshold parameter [28].

In our research we are trying to run complex algorithms on a low powered edge device in real time. We picked real time object detection algorithms to run on our edge devices [29]. Because object detection algorithms require extensive processing power and it has to be run on real time. So our goal is to run an object detection algorithm on an edge device.

We look though many object detection algorithms like Fast R-CNN(Convolutional Neural Network), Faster R-CNN, Single Shot Detector (SSD), YOLO [1], after we referring and studying all those object detection algorithms, we finally decided to use Yolo as our algorithms and trying to modify it and run it on edge level.

3.1 Why YOLO and how does it work ?

YOLO is extremely fast and accurate compared to other object detection algorithms [30]. Compared to other algorithms YOLO able to provide higher mAP(mean average precision) while consuming the same amount of computation power.

YOLO see the image globally when making predictions. Unlike sliding window and region proposal-based techniques, YOLO sees the entire image during training and test time. YOLO makes less than half the number of background errors compared to Fast R-CNN. YOLO uses the features in the entire image to predict the bounding box. And also it predicts the all the bounding box of all classes simultaneously. In YOLO it divides the image into n x n grids. If a center of an image falls into a particular grid then that grid is responsible for making the bounding box for that particular object. So each grid cell will predict the B bounding box and the confident scores for that box , confident score is how confident that image falls into a particular classifier. If nothing falls into a bounding box, then the confident score is zero. confidence score is defined as

$$Pr(Object) \times IOU_{predicted}^{true} \tag{3.1}$$

Equation 3.1 Confidence Score

IOU is Intersection over union between the predicted box and ground truth of the image box. Each bounding box consists of 5 predictions: x,y,w,h and the confidence. The (x, y) coordinates represent the center of the box relative to the bounds of the grid cell. The width and height are predicted relative to the whole image. Finally the confidence prediction represents the IOU between the predicted box and any ground truth box. Each grid cell also predicts C conditional class probabilities, $Pr(Class_i|Object)$. These probabilities are conditioned on the grid cell containing an object. We only predict one set of class probabilities per grid cell, regardless of the number of boxes B. At test time we multiply the conditional class probabilities and the individual box confidence predictions

$$Pr(Class_i|Object) \times Pr(Object) \times IOU_{predicted}^{true} = Pr(Class_i) \times IOU_{predicted}^{true} \tag{3.2}$$

Equation 3.2 conditional class probabilities and the individual box confidence predictions

which gives us class-specific confidence scores for each box. These scores encode both the probability of that class appearing in the box and how well the predicted box fits the object.

3.2 Methodological approach

As shown in the example (as seen in the figure 3.1) YOLO model detects this as a regression problem. It divides the image into an S × S grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an S × S x (B x 5 + C) tensor.

Here s = 7 and and bounding boxes are 2 , and let say we have 20 labelled classes so C = 20. Then our final reduction is a 7 x 7 x 30 tensor.

Even this kind of algorithm is very computation complex for an edge device. Even if we run an algorithm like this on an edge device it will not give real time results.

So Our first approach is to divide the computation power across multiple edge nodes in real time. In the current implementation of YOLO, they only run the algorithm in one single unit. (as an example in apple detection platform they run it on Nvidia Jetson board). So if we can distribute the computation workload [31] across the multiple

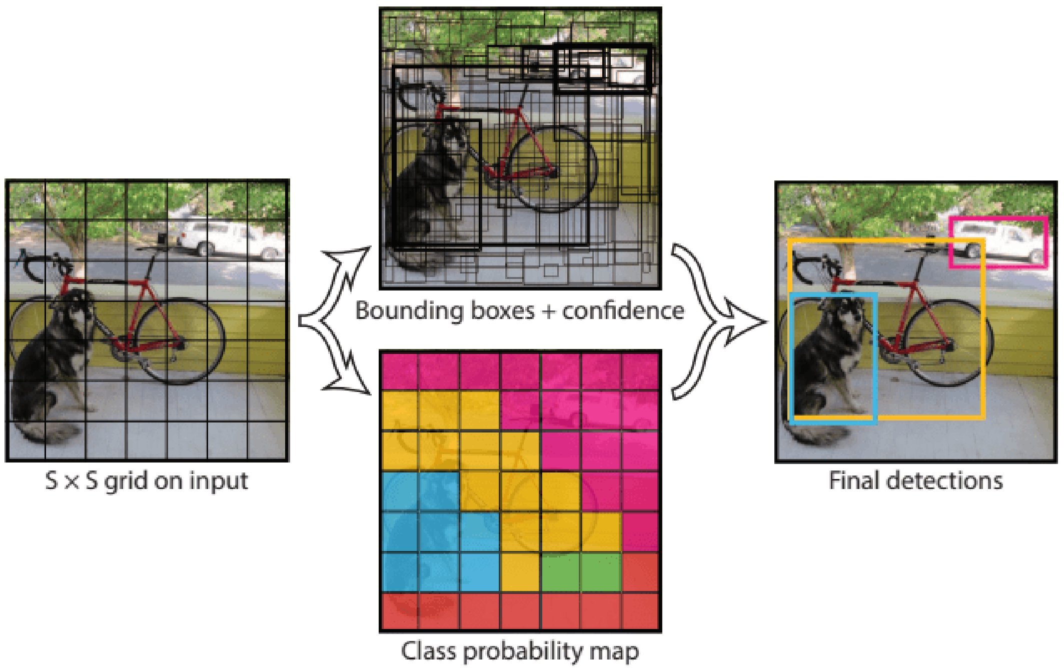

Fig. 3.1 Dividing the image into Grid and bounding boxes [1]

.

nodes (in multiple edge devices). Then we may able to archive real time results as multiple nodes will increase the computation power. We can archive this So by applying map reduce and distribute this in multiple low computational level devices(in multiple hardware devices). Using that method, we can use multiple lower powered devices to run the complex algorithms real time at edge level. Or we can distribute the multiple filters of the convolutional network across the multiple edge devices. Then we will be able to parallelize the computational workload in multiple nodes which again increase the computational power.

Our second approach is to reduce the computation cost of the algorithm by simplifying the final output tensor. We can achieve this by reducing the number of class probabilities and the image division grid of the YOLO.

Using the previous example if we use 2 class probabilities(human, non human) and 9 grids. Then the output tensor can be calculated using $S \times S \times (B \text{ x } 5 + C)$, where $S = 3$, $C = 2$ and $B = 2$ then the output tensor is 108, which is 92% reduction of the output tensor matrix.

3.2.1 Other

As in the above example it will only classify objects into human and non human categories. And minimizing the number of grids needs to be done with precise care. Because it will make it smaller, the area for a grid will be higher so computation for each grid will again rise up and it can cause a decrease in the accuracy of the final outcome. So the optimal number of grids will depend on the resolution of the image and the processing power of the devices we use. We have to run a test and find the optimal number which provides the most accuracy with minimal computational cost.

Chapter 4

Experimental Setup and Implementation

4.1 Prototype

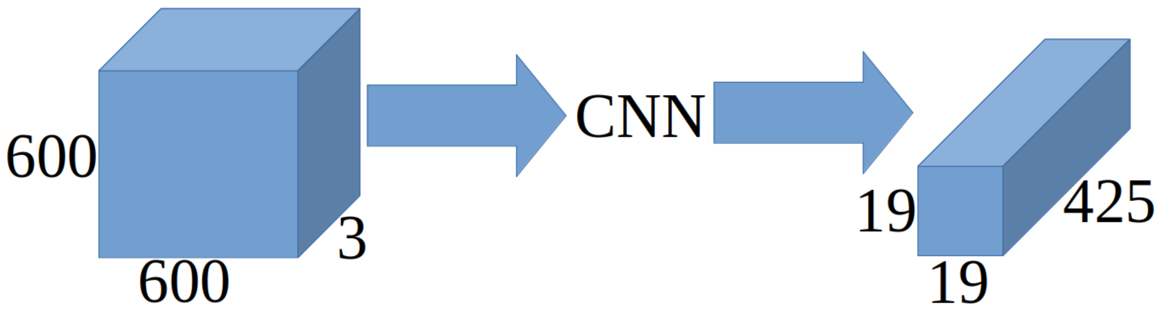

Fig. 4.1 CNN which takes 600x600x3 input and produce 19x19x425 output.

In this work we choose object detection as our use case which needs high computation power for the training and prediction phases. Raspberry pi computers are used as the end devices, Which runs the YOLO algorithm. YOLO algorithm is based on convolutional neural networks.Given an image, it feeds to an convolutional network and get an output based on partitioning make on the image and number class. Figure 4.1 shows a input and output example where 600x600x3 input is feed into the CNN where output is 19x19x425.In this example number of predicting classes are 80 while 5 anchor boxes are used for each grid cell.

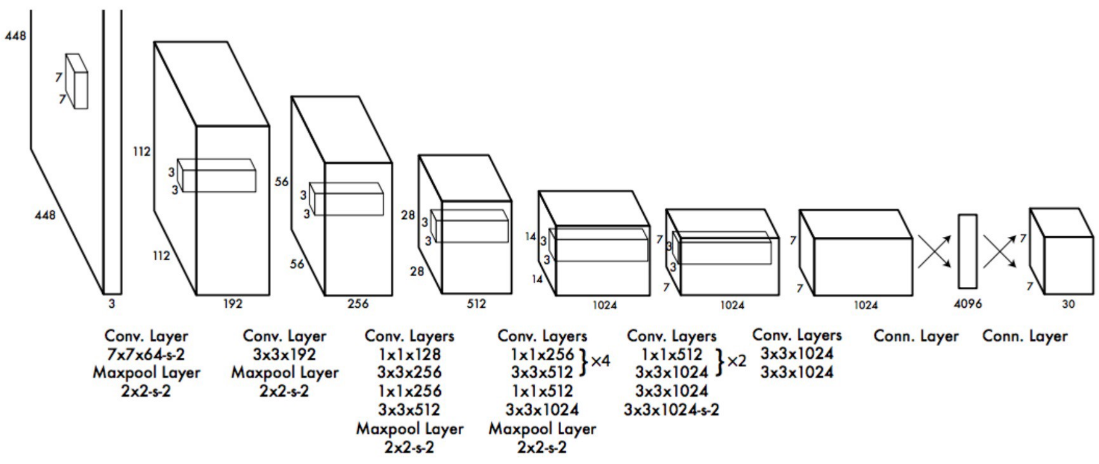

Fig. 4.2 YOLO architecture [1].

The convolutional neural network [32][33] consists of multiple filters in each layer so by dividing filters among multiple raspberry pi computers, the computation can be distributed and parallelized [31]. After computation done parallely then the output is merged and forward to the next layer.

Federated learning methods are used for the training network and object detection, Where each raspberry pi or a raspberry pi cluster trains a model based on local data and then the learned parameters are sent to higher layers (Fog, ROOF). Then the aggregation done on that layer to train a global model.

4.2 Research instruments

When it comes to object detection YOLO is a complex algorithm which consumes high computational resources [29]. DarkNet who are the founders of YOLO have trained a fairly large dataset using GPUs. Because the high complexity of the algorithm due to the complex Convolutional Network takes a very long time to train. To take an idea about the need of high resources given below is an instance of training COCO train 2014 (117,263 images) data set [2].

4.2.1 Resources used

Machine type preemptible n1-standard-8 (8 vCPUs, 30 GB memory)

CPU platform: Intel Skylake

GPUs: K80 ($0.14/hr), T4 ($0.11/hr), V100 ($0.74/hr) CUDA with Nvidia Apex FP16/32 HDD: 300 GB SSD

GPU	n	Batch size	img/s	Epoch time
K80	1	32x2	11	175min
T4	1	32x2	41	48min
T4	2	64x1	61	32min
v100	1	32x2	122	16min
v100	2	64x1	178	11min
2080Ti	1	32x2	81	24min
2080Ti	2	64x1	140	14min

Table 4.1 Training time for COCO data set under Cloud System [2].

The table 4.1 shows the statistical data of training the YOLO algorithm with GPUs which contains very high computational resources. The resources are allocated in the cloud environment. Even with such powerful resources available the time to train exceed 60 min at its best [2].

Our implementation targets to distribute the computation to edge devices with limited resources. We choose raspberry pi 3 boards as our edge nodes. Which have very limited resources compared to GPUs used in above scenarios. To distribute the computation among the nodes at the edge, we require a few raspberry pi boards for the cluster we instantiate the Debian OS on virtual machines. The environment is a virtually mounted Debian 10 operating system which is allocated 512 Megabyte of RAM and 1 GigaByte of Storage at the time of instantiation. This particular environment is a very constrained one compared to the GPU enabled cloud system as given above.

4.3 Data manipulation and Testing

For the testing we generated test data randomly. We have created a convolutional network which 256x256x3 image was given as input. Input image also generated randomly using numpy random methods. The input image has dimensions of 256x256x3. Filters and biases for each convolutional layer are also randomly generated.

```
np.random.seed(1)
image=np.random.randn(1, 256, 256, 3) #h256X256 image
W1=np.random.randn(3, 3, 3, 32)
b1=np.random.randn(1, 1, 1, 32)
W2=np.random.randn(3, 3, 32, 64)
b2=np.random.randn(1, 1, 1, 64)
W3=np.random.randn(3, 3, 64, 128)
b3=np.random.randn(1, 1, 1, 128)
hparameters1 = {"pad" : 129,"stride": 2}
hparameters3 = {"pad" : 65,"stride": 2}
hparameters2 = {"stride" : 2, "f": 2}
hparameters4 = {"stride" : 2, "f": 2}
hparameters5 = {"pad" : 32,"stride": 2}
```

Fig. 4.3 Weights and bias for testing.Randomly initialized weights and bias for each convolutional layer.

4.4 Pitfalls and workarounds

When the use case was to determine the AI algorithm as Object detection we went for YOLO [1]. It is a Complex convolutional network based algorithm. We had to learn how the Convolutional network works [32][33]. Then we went through how Convolutional Neural Networks are applied on object detection instead of basic Neural Network [30]. Meanwhile we learnt the map reduce techniques so we can apply that to distribute the computation [23]. Applying the Map reduce on ROOF nodes was a challenging task. Distributing the computation has to be done with several nodes. We tested the compatibility of running YOLO on different IoT devices. In our research we are using the raspberry pi as the edge node. We required several raspberry pi boards to process to use in our distributed architecture. As we are not possessing a few physical raspberry pi boards, we emulated them. Selecting the best emulator for the running Neural network was a challenge because the emulators have constraints which were from vendors. We had to try the three types of emulators for raspberry Pi. QEMU, VirtualBox with Raspbian and VMware with Raspbian Installed. We had to go through all the types of emulators as there were various problems that occurred with different softwares. Some problems we faced could be solved by referring to the Documentations and discussion forums and the Stackoverflow community. But when we were halfway through the implementation and then if a problem unique to the certain vendor occurs we had to change the emulator, which was time consuming and challenging. In the emulator we tested the existing YOLO implementations using tensorflow but the Tensorflow and Keras first. But those implementations does not compatible with the debian OS on virtual machine.

Chapter 5

Results and Analysis

5.1 Results

The implementation was run on three instances to measure the results.Initially the CNN of the YOLO algorithm was deployed on the colab cloud and tested the execution time.

The resource allocations are as follows

CPU RAM	GPU RAM
1.77GB	1.43GB

Table 5.1 Resource allocation.

Execution timing results

CPU	GPU
3.06066247199999 s	0.09349705900001481 s

Table 5.2 Execution timing results.

Then the YOLO on the Raspberry pi Virtual Environment Without parallelizing the convolutional network, to get an output it took 65531 ms.

After that Parallel implementation of YOLO distributing the Convolutional Neural Network was run on three Raspberry Pi boards simultaneously. The execution time was 32744 ms.

Environment	Colab CPU (1.43 GB RAM)	Colab GPU (1.77 GB RAM)	Non parallel Rasp-berry Pi	Parallel with three raspberry Pi
Execution time(ms)	3060.6624	93.4970	65531	32744

Table 5.3 Comparison of the Execution time of CNN of the YOLO algorithm.

5.2 Analysis

Based on the results it is evident that the object detection algorithm we choose, YOLO which has a complex Convolutional Neural Network performs well in the High resource enabled environment like Google Colab Cloud. In the Cloud environment the GPU and CPU which have capabilities upto 12 GB YOLO performs in separate efficiency. We ran our optimized YOLO algorithm in the High computationally capable CPU and GPU.

$$\frac{GPU\, Execution\, time(s)}{CPU\, Execution\, time(s)} = \frac{3.0606}{0.0934} = 32.76$$

Our YOLO Object detection algorithm performs nearly 32 times better than CPU in GPUs. At the next instance we ran the optimized algorithm in a single Raspberry Pi within the virtual environment which only has 512 MB of RAM at its max. Then the parallel implementation of YOLO which distributes the CNN was run on three Raspberry Pis Simultaneously. When the execution time in Colab cloud is compared with execution time is any raspberry Pi implementation Colab cloud performs well. Because of the availability of high computational resources. But the goal is to perform the complex CNN operations at the edge. So we have successfully deployed our YOLO implementation which consists of the optimized CNN, and performed the tasks with the very constrained and resource limited environment of the Raspberry Pi.

$$\frac{Resource\ availability\ in\ Colab\ Cloud - GPU(RAM)}{Resource\ availability\ of\ single\ Raspberry\ Pi(RAM)} = \frac{12\ GB}{512\ MB} = 23.43$$

The reason YOLO performs better at the Colab Cloud is that Colab Loud provides around 24 times better Computational resources compared to a single edge Raspberry Pi. But our implementation of YOLO with the optimised CNN performed within around 65531 ms even in the very resource constrained environment. Further the optimized YOLO algorithm was distributed to three parallel Raspberry Pis.

$$\frac{Parallel\ implementation\ execution\ time(ms)}{Single\ Raspberry\ Pi\ execution\ time(ms)} = \frac{32744}{65531} = 0.499$$

With the parallel implementation the Execution time is reduced nearly by a factor of two. This parallel implementation which distributes the computation among the Edge nodes performs better even with the very constrained and limited resources available at the edge devices.

Chapter 6

Conclusions and Future Works

The aim of the project is to enable real time processing at the edge with limited resources. So with combining above mentioned techniques real time processing at the edge is achievable. For this phase we have studied about different techniques which can apply in order to enable real time processing in IoT devices with limited resources. In this study we have developed a distributed CNN which can be used to implement YOLO. Furthermore, we propose a novel approach where end devices consisting limited resources can train and generate real time decision in distributed manner, where their computations are distributed among other multiple nodes or offload the computation to the upper layer when the resources run out. We have distributed the CNN over multiple Raspberry Pis. For this phase we have used a fixed size raspberry pi cluster. The performance of the CNN has been measured under different conditions and platforms. Based on the results it can be concluded that by distributing CNN over multiple nodes the computation latency can be reduced.

The CNN that we have implemented in this study can be optimized further. In future works our aim is to optimize CNN and distribute the CNN over dynamic cluster, then combine CNN with federated learning and policy technology to fine tune the parameters of the algorithm so that the YOLO algorithm can train and detect the objects in real time with limited resources at the edge while harvesting the advantages of a distributed architecture enabling the real-time processing at the edge.

References

[1] J. S. D. R. G. A. F. Redmon, "(YOLO) You Only Look Once," *Cvpr*, 2016.

[2] "ultralytics/yolov3: YOLOv3 in PyTorch >ONNX >CoreML >TFLite."

[3] M. Merenda, C. Porcaro, and D. Iero, "Edge Machine Learning for AI-Enabled IoT Devices: A Review," *Sensors*, vol. 20, p. 2533, apr 2020.

[4] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated Learning with Non-IID Data," 2018.

[5] W. Yu, F. Liang, X. He, W. G. Hatcher, C. Lu, J. Lin, and X. Yang, "A Survey on the Edge Computing for the Internet of Things," *IEEE Access*, vol. 6, pp. 6900–6919, 2017.

[6] G. I. Klas, "Fog Computing and Mobile Edge Cloud Gain Momentum Open Fog Consortium , ETSI MEC and Cloudlets," pp. 1–13, 2015.

[7] J. An, W. Li, F. L. Gall, E. Kovac, J. Kim, T. Taleb, and J. Song, "EiF: Toward an Elastic IoT Fog Framework for AI Services," *IEEE Communications Magazine*, vol. 57, no. 5, pp. 28–33, 2019.

[8] R. Mahmud, R. Kotagiri, and R. Buyya, "Fog Computing: A Taxonomy, Survey and Future Directions," *Internet of Things*, vol. 0, pp. 103–130, nov 2016.

[9] V. Mnih, A. Puigdomènech Badia, M. Mirza, T. Harley, T. P. Lillicrap, D. Silver, and K. Kavukcuoglu, "Asynchronous Methods for Deep Reinforcement Learning Volodymyr," *International Conference on Machine Learning*, vol. 48, 2013.

[10] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, pp. 3207–3214, 2018.

[11] T. Hester, T. Schaul, A. Sendonaris, M. Vecerik, B. Piot, I. Osband, O. Pietquin, D. Horgan, G. Dulac-Arnold, M. Lanctot, J. Quan, J. Agapiou, J. Z. Leibo, and A. Gruslys, "Deep q-learning from demonstrations," *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, pp. 3223–3230, 2018.

[12] X. Wang, Y. Han, C. Wang, Q. Zhao, X. Chen, and M. Chen, "In-edge AI: Intelligentizing mobile edge computing, caching and communication by federated learning," *IEEE Network*, vol. 33, pp. 156–165, sep 2019.

[13] M. Chen, H. V. Poor, W. Saad, and S. Cui, "Wireless Communications for Collaborative Federated Learning in the Internet of Things," pp. 1–17, 2020.

[14] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Transactions on Intelligent Systems and Technology*, vol. 10, no. 2, pp. 1–19, 2019.

[15] L. U. Khan, N. H. Tran, S. R. Pandey, W. Saad, Z. Han, M. N. H. Nguyen, and C. S. Hong, "Federated Learning for Edge Networks: Resource Optimization and Incentive Mechanism," pp. 1–7, 2019.

[16] J. Konečný, H. B. McMahan, D. Ramage, and P. Richtárik, "Federated Optimization: Distributed Machine Learning for On-Device Intelligence," pp. 1–38, 2016.

[17] H. Brendan McMahan, E. Moore, D. Ramage, S. Hampson, and B. Agüera y Arcas, "Communication-efficient learning of deep networks from decentralized data," *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017*, vol. 54, 2017.

[18] M. Villari, M. Fazio, S. Dustdar, O. Rana, and R. Ranjan, "Osmotic Computing: A New Paradigm for Edge/Cloud Integration," *IEEE Cloud Computing*, vol. 3, pp. 76–83, nov 2016.

[19] L. Lovén, T. Leppänen, E. Peltonen, J. Partala, E. Harjula, P. Porambage, M. Ylianttila, and J. Riekki, "EdgeAI: A Vision for Distributed, Edge-native Artificial Intelligence in Future 6G Networks," tech. rep.

[20] Y. Hao, Y. Miao, L. Hu, M. S. Hossain, G. Muhammad, and S. U. Amin, "Smart-Edge-CoCaCo: AI-Enabled Smart Edge with Joint Computation, Caching, and Communication in Heterogeneous IoT," *IEEE Network*, vol. 33, pp. 58–64, mar 2019.

[21] P. Mach and Z. Becvar, "Mobile Edge Computing: A Survey on Architecture and Computation Offloading," *IEEE Communications Surveys and Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

[22] Y. Liu, J. Yang, Y. Huang, L. Xu, S. Li, and M. Qi, "MapReduce Based Parallel Neural Networks in Enabling Large Scale Machine Learning," *Computational Intelligence and Neuroscience*, vol. 2015, 2015.

[23] M. Chen, W. Wang, S. Dong, and X. Zhou, "Video vehicle detection and recognition based on mapreduce and convolutional neural network," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10942 LNCS, pp. 552–562, Springer Verlag, 2018.

[24] S. B. Calo, M. Touna, D. C. Verma, and A. Cullen, "Edge Computing Architecture for applying AI to IoT," tech. rep.

[25] Z. Zou, Y. Jin, P. Nevalainen, Y. Huan, J. Heikkonen, and T. Westerlund, "Edge and Fog Computing Enabled AI for IoT-An Overview," tech. rep.

[26] X. Masip-Bruin, E. Marin-Tordera, A. Jukan, and G. J. Ren, "Managing resources continuity from the edge to the cloud: Architecture and performance," *Future Generation Computer Systems*, vol. 79, pp. 777–785, feb 2018.

[27] K. Bilal, O. Khalid, A. Erbad, and S. U. Khan, "Potentials, trends, and prospects in edge technologies: Fog, cloudlet, mobile edge, and micro data centers," *Computer Networks*, vol. 130, pp. 94–120, jan 2018.

[28] S. Taherizadeh, V. Stankovski, and M. Grobelnik, "A capillary computing architecture for dynamic internet of things: Orchestration of microservices from edge devices to fog and cloud providers," *Sensors (Switzerland)*, vol. 18, sep 2018.

[29] V. Mazzia, A. Khaliq, F. Salvetti, and M. Chiaberge, "Real-time apple detection system using embedded systems with hardware accelerators: An edge AI application," *IEEE Access*, vol. 8, pp. 9102–9114, 2020.

[30] K. Jo, J. Im, J. Kim, and D. S. Kim, "A real-Time multi-class multi-object tracker using YOLOv2," *Proceedings of the 2017 IEEE International Conference on Signal and Image Processing Applications, ICSIPA 2017*, pp. 507–511, 2017.

[31] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” tech. rep.

[32] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, “A convolutional neural network for modelling sentences,” *52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014 - Proceedings of the Conference*, vol. 1, pp. 655–665, 2014.

[33] M. E. Paoletti, J. M. Haut, J. Plaza, and A. Plaza, “A new deep convolutional neural network for fast hyperspectral image classification,” *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 145, pp. 120–147, 2018.