POLITECNICO DI TORINO

Master's Degree in Electronic Engineering

Master's Degree Thesis

Offloading for Efficient Execution of AI applications in Edge Computing

Supervisors

Prof. Jaya Prakash CHAMPATI (IMDEA)

Prof. Luca VASSIO (POLITO)

Candidate

Rishi TRIPATHI

September 2023

Summary

With the dawn of powerful and energy-efficient hardware in the low-power computing space, there is the need to make a decision to go either for performance or for efficiency. Although not completely exclusive to each other there is usually a winner during practical applications. However, the chosen solution might not really hold up once the energy constraints show up in the form of limited power sources for mobile devices.

The performance capabilities of these edge devices are severely limited to thermal and energy constraints. For the sake of improving efficiency and accuracy in machine learning and running neural networks on board such portable devices we are faced with the following simple yet brutal choice. The first option involves wasting local resources in the form of energy and computational power for a low-accuracy output. The second option is sending (offloading) the job to be processed onto a remote server with high accuracy but incurring massive energy penalties on the local level in the form of energy wasted for transmission and at the server in the form of the energy used to compute the result.

This work deals with how to optimise that decision so that we can optimise for accuracy and energy simultaneously with a sharper focus on energy. The Problem Statement for this research project is to develop a regime or algorithm that can help us make the optimum decision to offload the task from an edge device to a server so that we can improve upon the energy budget and average accuracy across all jobs.

In order to measure the performances and answer the research questions, e have used two devices over a wireless [WLAN] network. The first device is a low-power (6.5W) Raspberry Pi 4 which represents our local device or edge device. On this device, a small version of the MobileNetV2 image classifier is running. The second device is a computer as a server which is high power(120W) and it is running a larger version of MobileNet V3 image classifier which is more accurate. We are measuring the energy consumption of both these devices with the use of external measuring devices. The Raspberry Pi has a set of images (jobs) on its local memory that it has to classify. Before each image can be classified it must be reshaped into 224x224 pixels. We have an energy budget that is the sum of energy required to

reshape all jobs and classify all jobs on the local device. The local device has three options for each image, it can either reshape and classify itself paying the associated energy costs, it can transmit to the server for reshaping and classification by paying the transmission cost and finally it can reshape the image and then transmit to pay the reshape energy cost to save on transmission energy costs for larger files. Our goal is to somehow make the best decision out of the three to finally consume less energy than our energy budget while simultaneously improving accuracy. This decision is made based upon the expected values of these different energies for each image, using lookup tables that have been created by data collection, essentially making this a computing time(energy) to space trade-off.

The Implemented solution was run on three sets of data from the ImageNet database and it allows us to improve upon the energy budget by 1.93 percent in our best case and an accuracy gain of .42 percent in top 1 accuracy. The other two sets that have been tried stand to gain .36 percent in accuracy and 1.26 percent in energy budget for set 2 and .22 percent for accuracy and .89 percent in energy budget in set 3. Since there is not much focus on research in the field of studying energy consumption in general for networks and edge devices these results reveal to us the possibility of squeezing water out of rocks when it comes to saving energy. If the data being processed is favourable and more uniformly distributed in a certain size range, more than 5-10 percent of energy savings can be made.

Acknowledgements

I would like to express my deepest gratitude to my thesis guides Prof. Luca Vassio and Prof. Jaya Prakash Champati for supporting me through the most tumultuous time in my life, their guidance and wisdom and the ability to not give up on me even when i was not sure of myself. I would like to also mention the phenomenal guidance and help they provided me on every step of this research project. Professor Champati helped me by suggesting me texts and courses to fill gaps in my knowledge and providing continued guidance during the whole process of experimentation, egging me on in the right direction whenever i wandered off on a tangent. Professor Luca taught me some mathematical tricks that helped me reduce computational complexity during the development of the algorithm. His general help and support finally brought me back on track in my life and for that i am forever grateful

I would also like to thank my brothers Sachit , Shashwat and my uncle Akhilesh Kumar Tripathi for always being there in need.

I am lucky to have worked under and with these splending individuals and i will be forever grateful for their contribution in my project and the impact they had on my life.

Table of Contents

List of Figures

List of abbreviations and symbols

Symbols used

j -> one job from the set of all available jobs J
i -> 0 for local machine, 1 for server
a_i -> accuracy of model i
x_j -> if job j is assigned to the local machine , $x_j = 1$ and $x_j = 0$ if job j is assigned to the server
E_{ij} -> energy required by machine i to perform job j
R_{ij} -> energy required to reshape the job j on machine i
T_j -> transmission energy required to send job j to the server
T_0 -> transmission energy required for a reshaped job
y_j -> decision variable that is 0 if reshaping of the job is done locally, 1 if it is done on the server
N_{rg} -> Max available energy/ Energy constraint
E_{alg} -> Energy consumed by the machine 0 to run the algorithm.
IOT -> Internet Of Things

Chapter 1

Thesis Objective and Contributions

1.1 Introduction

The problem of offloading computation is much older than one might think and present at many levels of abstraction. There has always been the case of distributed computing systems and as always there is the scheduling problem for multi-core systems. At different levels of abstraction these problems are dealt with in very different manners but at the end of the day all of these approaches are at their heart similar because the system gets optimised for a certain attribute with a constraint on other available parameters. This constraint in our case is twofold, one of energy and the other of accuracy.

The problem that we have at hand requires us to optimise the *energy consumption and accuracy* of an image classification model running at a resource-constrained device with the option of offloading that task to a more powerful classifier running on a server that is not resource-constrained.

The setup that we have chosen is one in which a set of N images present on our remote resource-constrained device which has the capability to perform three functions, (1) classification model (ml) which helps classify the incoming image into its category, (2) a reshaping function on it which reshapes the current image to a fixed size that is required as input by the classification model and (3) finally it can send data to the server, all of these functions have associated energy costs that vary from job to job. The server also has the same functionality as the resource-constrained device albeit better. However, the caveat here is that though

all the functions are costlier in terms of energy to perform at the server. The limited energy at the local resource-constrained device is far more precious compared to the server which does not draw power from an energy-constrained container like a battery.

1.2 Motivation

The motivations behind developing such an algorithm are the following:

1. Extended battery life on mobile devices
    - *Heading* Every mobile device has an energy limitation which is due to the fact that it has a battery, which is a limited power source. Similar devices like wireless sensors that are used to collect data run on batteries that are usually solar-powered, energy management in these cases is of the utmost importance. Every joule of energy that can be conserved for these types of devices contributes greatly to the performance in the long term.
    - *Decreased Hardware Degradation* - Batteries are usually rated for a certain number of charges and recharges. Once you hit the limit on the number of charge-discharge cycles the performance and charge-holding capacity of the battery goes down. With the help of this algorithm, we can increase the amount of work that can be done in the given battery capacity thereby increasing the total amount of work done in the life cycle of the battery by the device. If the amount of work to be done is limited every day the frequency of the battery requiring a recharge will be lower and therefore the life of the battery will go up depending upon the workload.
2. Benefits in other distributed computing systems
    - By using a generalized version of this approach for other distributed computing systems with resource-constrained nodes one can improve and optimize the use of the constrained resource while saving energy in the process.
3. Environmental Benefits
    - The world is headed to an era of IoT (*Internet of Things*). Small devices and sensors with batteries will be more and more common. For example, devices like smartwatches can make use of similar algorithms for their specific purposes like measuring body data.

One device saving twenty joules is not a lot but the number of such devices is bound to increase in the world over time as more and more people begin to use more and more digital devices. Twenty joules per device for a million devices adds up to a substantial number and compounding that over the lifetime of the device it will be a net positive for the world. Since the energy demands for the world are always rising this might add a drop in the bucket of energy that we will need in the future.

1.3 Thesis Structure

This thesis is organized as follows. Section 2 illustrates the literature review done for the project studying relevant work in the field. Section 3 formalizes the problem also encapsulating the different approaches to modelling the problem some of which panned out and others not so much. We delve into the why and how of different approaches. Section 4 describes the data-sets that have been used and how the data was prepared. Section 5 presents the methodologies, devices, models and other things that were used in the setup for the experiment. Section 6 summarizes the main experimental results. Finally, Section 7 draws conclusions and summarizes the future research agenda.

Chapter 2

Literature Review

Energy efficiency is a paramount concern in the realm of (IoT) due to the proliferation of battery-powered and resource-constrained devices. The need to maximize the lifespan of these devices while maintaining their functionality has prompted extensive research in the field of energy-efficient computing. One promising technique to achieve energy savings is *task offloading*, which involves delegating computationally intensive tasks to more capable devices or cloud resources. This literature review aims to provide an overview of energy-efficient computing in IoT and explore the role of task offloading in addressing energy constraints.

To structure the literature review effectively, we will segment this section of the chapter into distinct sections. These sections will provide insights into previous research related to the following aspects: (1) Energy-Efficient Computing in IoT (2) Task Offloading Strategies (3) Algorithms and Decision-Making for Offloading (4) Offloading in Resource-Constrained Devices (5) Gaps in Current Literature (6) Conclusion

2.1 Energy-Efficient Computing in IoT

Energy efficiency stands as a foundational principle in IoT design, as many IoT devices rely on limited battery capacities. Dynamic voltage and frequency scaling (DVFS)[1], low-power processors, and sleep modes are some of the techniques that have been employed to reduce power consumption. These methods aim to strike a balance between performance and energy efficiency, adapting device behavior to the current workload. However, since we are focused on the implementation of an offloading decision making algorithm, many of the tips and tricks that we

gain from the insight granted by studying this research would not be of much use since we are not studying the running of a system for an extended period of time, however some of the insights help us set up our remote device to be more efficient.

2.2 Task Offloading Strategies

Task offloading is a key strategy to improve energy efficiency in IoT and edge computing environments[2]. It encompasses both computation offloading (offloading parts of a computation) and data offloading (sending data to remote servers for processing). One of the primary advantages of task offloading is its potential to reduce energy consumption by allowing less capable devices to offload resource-intensive tasks to more powerful devices or cloud servers . This approach can lead to improved response times and better resource utilization. This insight allows us to visualise what strategy might or might not work in our case. A key strategy that comes to mind when is partial processing of the workload before it is completely offloaded to the server for processing in order to save energy during the process of transmission.

2.3 Algorithms and Decision-Making for Offloading

The effective implementation of task offloading relies on sophisticated algorithms and decision-making processes. Research has explored heuristic-based approaches[3], machine learning algorithms, and optimization techniques for making informed offloading decisions. These algorithms consider various factors, including task characteristics, device capabilities, and network conditions, to determine whether and what to offload. The performance of different offloading algorithms is evaluated in terms of energy savings and latency reduction. Since we are solely focused on optimising for energy we will concentrate on the energy aspect of the optimisation techniques.

2.4 Offloading in Resource-Constrained Devices (e.g., Raspberry Pi)

Resource-constrained devices, such as the *Raspberry Pi*, present unique challenges and opportunities for task offloading. Researchers have investigated the trade-offs between offloading tasks to more powerful devices or cloud resources in heterogeneous networks and executing tasks locally [4]. This balance is critical in scenarios where device capabilities and network bandwidth are limited. We aim to explore optimizations specific to IoT platforms like the Raspberry Pi, aiming to maximize energy savings without compromising performance and for this purpose studying such research is important in order to figure out things that we can try that others have not. The setup and aim of each network and node configuration is different so a mostly unique solution is required in every case. However it is better to know the types of solutions that end up working most of the time.

In some studies the subject has been studied[5] but since tracking energy consumption is a difficult task results are definitely present but rearely quantified. Key findings from these studies, along with their methodologies and novel contributions, provide valuable insights into the practical implications of offloading strategies.

2.5 Gaps in Current Literature

While substantial progress has been made in the field of energy-efficient computing and task offloading in IoT, certain gaps and challenges remain. There is a need for further exploration of unresolved research questions, including issues related to network latency, security, and real-time offloading decisions. Additionally, the scalability and adaptability of offloading strategies in complex IoT ecosystems warrant continued investigation. In the arms race of computing power on edge devices and the eventual death of Moore's law it is time to start paying more attention to energy consumption of such devices and how to optimise the situation for every last joule of energy.

2.6 Conclusion

In conclusion, energy-efficient computing and task offloading are integral components of IoT device management, enabling the conservation of valuable energy resources. The literature reviewed in this section underscores the significance of task

offloading as a means to achieve energy savings while maintaining the functionality and performance of IoT devices. This body of research informs the context and motivation for the current study, emphasizing the relevance of energy-efficient computing in IoT and the role of task offloading as a promising strategy.

Chapter 3

Problem Modelling

In this chapter, we will discuss the initial models along with their constraints. Based on these constraints, we will outline the evolution of our approach to address the limitations of the original model.

3.1 What is a Queue Model

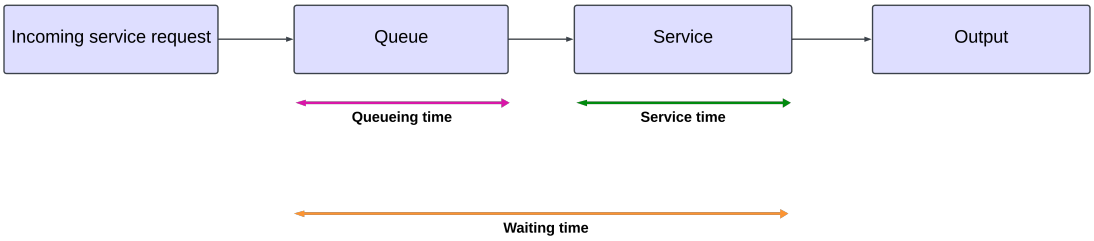

Figure 3.1: Sequential flow of a standard Queue Model.

A queue model, often referred to as a *queuing model*, is a mathematical and analytical framework used to study and understand the behavior of waiting lines or queues in various systems. These systems can be found in a wide range of applications, including customer service centers, manufacturing processes, computer networks, transportation systems, and more. Queue models help in predicting and optimizing the performance of such systems by analyzing the dynamics of entities waiting for service or processing.

A standard queue model follows a sequential flow that involves several key elements:

Arrivals: In a queue model, entities arrive at the system for some service or processing. These entities can represent customers at a service center, packets in a computer network, or any other relevant items. Arrivals can occur at random intervals or follow a specific pattern. You can see Arrivals in the figure 3.1 as the incoming service request.

Queue: When entities arrive at the system but cannot be immediately served or processed, they join a waiting line or queue. This queue holds entities in the order they arrived, following the "first come, first served" (FCFS) principle, although variations like priority queues exist.

Service: Entities are served or processed by one or more service points or servers. The service time for each entity can also vary; it may be constant or follow a probability distribution.

Departures: After receiving service, entities leave the system and are considered "departures." The departure rate is influenced by the service rate and the number of servers available. The Departures are represented in figure 3.1 as output.

Queue Length: The queue length represents the number of entities waiting in the queue at any given time. It is a critical metric for understanding system performance.

Utilization: Utilization measures the fraction of time that servers are busy serving entities. It is calculated as the ratio of the arrival rate to the service rate and provides insights into the efficiency of the system.

Waiting Time: The waiting time for an entity in the queue is the time elapsed from arrival until it begins to receive service. Minimizing waiting time is a common goal in queue model analysis.

These basics will serve as a foundation for our purpose and we can build our own setup in its image, easily facilitating the formation of equations.

3.2 History of Queue Model

The entire discipline of telephone traffic engineering as well as queuing theory were both developed by the Danish mathematician, statistician, and engineer *Agner Krarup Erlang*, who oversaw a technical lab at the Copenhagen Telephone Co. in the early 20th century. Telephone companies widely accepted his substantial

research on wait times in automated telephone services and his recommendations for more effective networks. In 1917, he solved the M/D/1 queueing model, and in 1920, the M/D/k (Kendall Notation) queueing model used a *Poisson process* to simulate the quantity of phone calls arriving at an exchange. Here, the letter M, stands for "Markov" or "memoryless." D stands for "deterministic," meaning jobs entering the queue require a definite quantity of service, and finally k denotes the number of servers at the queueing node (k = 1, 2, 3,...) . Jobs will queue up and wait for service if the node has more jobs than servers. Also in 1930, Felix Pollaczek's proposed the solution to the M/G/1 queue was later reformulated in probabilistic terms by Aleksandr Khinchin, and is now referred to as the *Pollaczek-Khinchine* formula.

Mathematicians started to become interested in researching queueing theory after the 1940s. For example, in 1953, David George Kendall solved the GI/M/k queue and developed what is currently referred to as Kendall's notation, a modern notation for queues. Pollaczek investigated the GI/G/1 in 1957 using an integral equation. John Kingman provided what is now known as *Kingman's method* to calculate the average wait time in a G/G/1 line. Then in early 1970s, Leonard Kleinrock worked on the application of queuing theory to message switching and packet switching, respectively. His doctoral dissertation at the Massachusetts Institute of Technology in 1962, which was later published as a book in 1964, was his debut contribution to this discipline. His theoretical work, which was published in the early 1970s, served as the theoretical foundation for packet switching on the *ARPANET*, the predecessor to the Internet.

Subsequently, researchers began examining the utilization of queues featuring inter-arrival and service time distributions modeled as phase-type distributions. These investigations encompassed the application of both the matrix geometric technique and the matrix analytical method. Now in the modern context of wireless networks and signal processing, systems with linked orbits play a significant role in queueing theory. The queueing theory is now widespread in other domains i.e. product creation where (material) products have a spatiotemporal existence, in which they have a volume and a duration for example food at restaurants. However, performance measurements for the M/G/k queue is one unresolved issue.

3.3 Usage of Queue Model

If we look around us, we find an abundance of queues in our daily lives, from the line at the bank to the line at the supermarket. The number of queues a person has to deal with in a lifetime is staggering and no one really thinks about it. Whenever

there are not enough resources, queues can form. Every company can tolerate some queues because a complete lack of queues would indicate an expensive overcapacity. So the goal of queuing theory is to create efficient, cost-effective systems that can serve customers promptly and effectively.

Queuing theory is used in supermarkets to speed up checkout procedures. By taking into account elements like the time of day and season, they compile data on client arrival rates and cashier service rates. Mathematical models, such the *M/M/1 queue model*, can be used to forecast client wait times and line lengths. The number of checkout lanes to open or close during the day is decided using this information. Additionally, express lanes for speedy service and self-checkout alternatives are available at supermarkets. A queue can be managed effectively by using separators, clear signage, and displays of expected wait times. By matching employee scheduling to demand patterns, over-staffing during calm times is avoided. In order to provide a seamless and effective purchasing experience, constant process changes are informed by customer feedback.

Similarly in the case of manufacturing and supply chain management Queueing theory is applied extensively. It supports production scheduling, workstation optimization, quality assurance, and inventory management in manufacturing. It helps manufacturers manage inventory effectively while streamlining processes, removing bottlenecks, and maintaining product quality. Queueing theory helps with demand forecasting, supplier selection, logistics and transportation optimization, warehouse management, and risk assessment in the context of supply chain management. It guarantees that products are delivered from suppliers to clients in an expedient manner, cuts down on transportation expenses, and optimizes warehouse operations.

Additionally, it is essential for inventory management, assisting businesses in determining reorder points and safety stock levels. This guarantees that goods are accessible when needed without incurring excessive inventory costs.

Businesses can use queuing theory to make data-driven decisions, increase operational effectiveness, cut expenses, and ultimately deliver goods to customers quickly and affordably.

With these two examples it becomes abundantly clear that even though queuing theory is applicable in both these scenarios they are not exactly the same in setup and execution. This tells us that each use-case of the queuing theory requires a slightly different approach and we get a different set of equations for different situations.

3.4 The First Queue Model

Objective: The problem that we are discussing encompasses a distributed system with a small less accurate system which is a local machine and an absolutely accurate system which we refer to as the server. We have a slew of jobs that need to be performed and the basic question here is how to allot the jobs to the machines such that their individual input queues remain stable (which means they neither blow up nor starve) and we receive an optimum scheduling that gives us the best possible use of our computational budget.

Problem Formulation: To articulate the problem, we start with the *first model*, we try to adopt was supposed to be purely mathematical and with a mathematically obtainable solution, which would later be verified with experimentation.

For this first model, we decided to model the problem as follows. Since we had a local machine and a server we used a probabilistic model to represent our case. Some things to take note of are that the local machine has an accuracy of p and therefore there is a chance that it will fail to do the job with the probability of $1 - p$, the server is assumed to be capable of doing the job with a probability of 1. the only difference is the time required, the local machine can process the job in a time of 1 and the server needs t_f time to process the job where $t_f > 1$. The simplest explanation of the model is in the image.

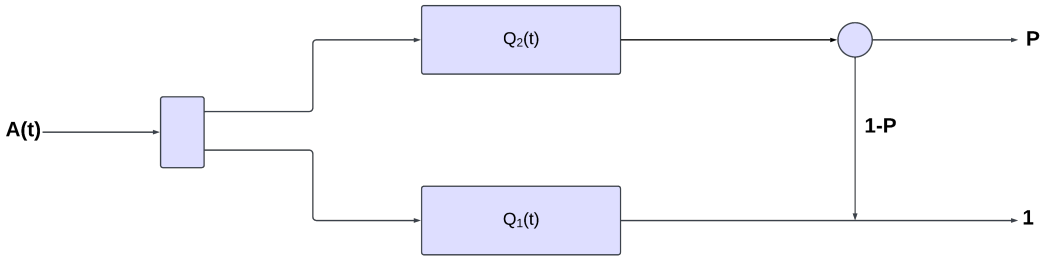

Figure 3.2: Model of the Problem Using Queues

Where $A(t)$ is a random process (a random process is a collection of random variables usually indexed by time or sometimes by space) the random process $A(t)$ was supposed to be a representation of the jobs randomly arriving for processing in the system. The queue Q1(t) is the place where all the arriving jobs are being queued for processing on the local machine.

The queue Q2(t) is the place where all the arriving jobs are being queued for sending to server.

$$A(t) = \begin{Bmatrix} 1 & k \\ 2 & 1-k \end{Bmatrix} \tag{3.1}$$

$$\alpha(t) = \begin{Bmatrix} A(t)=1 & Q_1 & Q_2 & \\ A(t)=2 & 2Q_1 & 2Q_2 & Q_1, Q_2 \end{Bmatrix} \tag{3.2}$$

$\alpha(t)$ which can make one of the following 3 decisions which are to

1. *Process Locally* - In this cast the job would be forwarded to another independent process $s_1(t)$ which would classify the job correctly with a probability of p1 if the classification was wrong then it would feed the job to $s_2(t)$

2. *Send to server* - this would entail invoking $s_2(t)$ which would then send the job to queue $Q_2(t)$ which would be used to transmit the job to server. take note that if a fail would happen in the process $s_1(t)$ the jobs from there would ultimately end up here.

3. *Idle* - This would be invoked in case the queue $Q_1(t)$ is empty.

$$Q_1(t+1) = max[Q_1(t) - b(t),0] + a(t) \tag{3.3}$$

3.3 is the general equation for the next state of a queue where a(t) is arrivals in the queue and b(t) is departures. Our specific cases are listed below. We are using the Lyapunov function f to check our queues.

$$Q_1(t+1) = max[Q_1(t) - p + f_1(A(t), \alpha(t))] \tag{3.4}$$

$$Q_2(t+1) = max[Q_2(t) - (1-p)) + f_2[(A(t), \alpha(t)] + (1-p)] \tag{3.5}$$

$$Q_1(1) = max(0-0) + f_1(A(t), \alpha(t)) \tag{3.6}$$

for $A(t)$ to be maxed we have 3 options $2Q_1$, $2Q_2$, Q_1Q_2

current policy $\rightarrow$ if 1 then send to Q_2 if 2 send to Q_1, Q_2.

$$Q_1(t) = 0 + f_1\overline{(A(t), \alpha(t))} \tag{3.7}$$

$$Q_1(t) = 0 + p \tag{3.8}$$

$$Q_1(2) = (p-1,0) + f_1\overline{(A(t), \alpha(t))} \tag{3.9}$$

its apparent that if $f_1\overline{(A(t), \alpha(t))} = p$ Q_1 can not build up

$$Q_2(1) = 0 + 1 \tag{3.10}$$

$$Q_2(2) = (1 - 1) + 1 + (1 - p) \tag{3.11}$$

$$Q_2(3) = (1 + 1 - p - 1) + 1 \tag{3.12}$$

in this case whenever there is a drop Q_2 can't stop blowing up

Queue State upon action

1) If queue 1 is populated and the action process locally is taken, the queue is reduced by 1 in the current slot with probability p

2) If queue 1 is populated and sent server action is taken then q1 is reduced by 1 and queue2 is increased by 1

3) If queue 2 is populated and the action to transmit to the server is taken then after t_f amount of time the length of q2 is reduced by 1

4) If queue 1 is empty then idle is chosen

5) If queue 2 is empty idle is chosen

Now this model though appearing genius is riffled with flaws. For one, the equations get complicated and monstrous as it involves *Lyapunov optimisation* for the queues which refers to the the use of a Lyapunov function to optimally control a dynamical system. Lyapunov functions are used extensively in control theory to ensure different forms of system stability. The state of a system at a particular time is often described by a multi-dimensional vector. A Lyapunov function is a non-negative scalar measure of this multi-dimensional state. Typically, the function is defined to grow large when the system moves towards undesirable states. System stability is achieved by taking control actions that make the Lyapunov function drift in the negative direction towards zero. In our case, we need to avoid the queues getting filled so that the solution can be generated in a favorable amount of time and the system does not drift to instability. Once we check for the Lyapunov drift for both our queues we find that the problem if solved for a solution always shows that the send to server option is never chosen by the process alpha and the population of the q1 and q2 are always in the ratio of p: 1-p which is due to the fact that the failed cases of the local machine end up on the server anyway. Thus though we did a bunch of mathematics the verdict remains that the solution in such a case is trivial and comprises of spamming the local machine with jobs with the fails passing to the server.

3.5 The problems with the Queue Model

The most intriguing aspect of this model revolves around the notion that the local machine is aware of its failures in a task, particularly in scenarios like image recognition where the veracity of classification results remains uncertain. Consequently, determining how to transmit these failures to the server without prior verification poses a significant challenge. Furthermore, any necessary verification must be carried out with absolute precision (100 percent), a quality only achievable throughe the server.

3.6 Insights gained From the Queue Model

The insight that we gain from this sudden brick wall we have encountered will be three-pronged.

1. . A remote system should have an energy budget which is a glorified version of a computational budget but with far more real world significance than a computational budget could convey.
2. The problem with accuracy of the local machine and server in doing the jobs necessitates optimising for accuracy as well since we never know when a wrong result is produced.
3. Energies for transmission and processing suddenly become relevant as we have to take into consideration energy budget as well.

3.7 Redefining of the problem parameters in a Real World Scenario

From what we understand here we need to be more realistic and practical with our model. So we need to Redefine the problem itself so that we can be more precise with our endeavors.

Our scenario therefore is going to be a set of two devices which are the server and the local device. The local device is an independent computer with an attached sensor/camera that sends it images periodically, these images are random in size and shape and quality, these images need to be classified into their correct categories, for this particular purpose the device will be running a small version of an image

classification ML(*Machine Learning*) model which is not too power intensive and has a certain defined accuracy. This local device is under an energy budget limitation, lets say it has a battery that is charged by solar panels and it does its job in the night after a full day of data collection with the battery fully charged initially. The images that the local device captures first need to be reshaped to a certain size so that they can be fed to the classification model running on the device, this process of reshaping also consumes energy. The process of transmitting the results to the server also takes nominal energy but the process of sending images to the server will cost energy too. The model that runs on the local machine will also consume a certain amount of energy per classification. The server has no such energy restraints and has a very high accuracy of prediction as the model running on it is not bound by computational power and energy budgets like the local machine has. The server can essentially do all the things the local machine can with no regards to power consumed.

3.8 The Final Model

The model constitutes of a *Low Power Device* (referred to as the local device) with an independent but limited power source and limited computation potential, A server (referred to as the server) with a high computing power and no limit on power, for our purposes this difference in computing power equates to a difference in accuracy of the classifiers with the server having an average accuracy of a_2 and the low power device with an average accuracy of a_1 these are linked up via a dedicated network which is wireless between the low power device and the server. There are N number of images that are present at the low power device which are to be processed and classified, each of these images are referred to as jobs . The low power device has a limited power supply which has a fixed amount of energy N_{rg}) available.

There are three possible operations that can happen on the low power device,

- Reshaping
- Prediction
- Transmitting image

There are three possible operations on the server

- Reshaping

- Prediction
- Receiving image

Each of these operations have an associated energy cost. With E_{ij} being energy required by machine i to classify job j, R_{ij} being energy required to reshape the job j on machine i, T_j is the transmission energy required to send job j to the server. Where i is a binary variable that assumes the value 1 for the local machine and 2 for the server. With these three possible operations available on both our devices we need to meet our goal of classifying the image. Classifying the image requires it to be reshaped into a dimension that can be fed to the classifier, Passing it through the classifier and recording the result. Now with the operations that are available to us we have 3 possible ways of getting the result,

- Reshape on Local -> Classify on Local,
- Reshape on local -> Transmit to server -> Classify on server
- Transmit to server -> Reshape on server -> Classify on server

With each image the reshaping and transmission energies depend upon the size and data contained within the image,there are certain caveats associated with the reshape function including the fact that it is cheaper in terms of energy to reshape the image if it needs to be up scaled than down scaled. This leads to the reshape function energy requirement to not scale linearly with the size of the file, The energy consumed for transmission scales perfectly linearly with the increase in file size. We have to increase the average accuracy of the classification while obeying the energy constraint. To introduce a mathematical model for the above problem we have to enlist the help of a couple of decision variables x_j if job j is assigned to the local machine , $x_j = 0$ and $x_j = 1$ if job j is assigned to the server. y_j -> decision variable that is 0 if reshaping of the job is done locally, 1 if it is done on the server

$$\sum_{i=1}^{2} x_j = 1 \forall j \in J \tag{3.13}$$

$$\sum_{i=1}^{2}\sum_{j=0}^{n} E_i.x_j + \sum_{j=0}^{n}\{(1-y_j)R_{1j} + y_j R_{2j}\} + \sum_{j=0}^{n} x_{2j}\left(y_j.T_j + (1-y_j)T_0\right) \leq N_{rg} \tag{3.14}$$

$$\sum_{j=0}^{n} E_1 x_{1j} + R_{1j}x_{1j} \leq N_{rg} \tag{3.15}$$

$$x_j \in \{0,1\}, y_j \in \{0,1\}, \forall i \in \{1,2\}, j \in J \tag{3.16}$$

$$A = \sum_{i=1}^{2} \sum_{j=0}^{n} a_i x_j \tag{3.17}$$

of course our goal here is to maximize A with respect to all constraints given in equations 1,2,3,4 now we can see that in equation 2 we have a product of two decision variables which makes this equations exponentially more difficult to solve with increase in the number of jobs therefore to simplify equation 2 and arrange the terms into a form that grants us more insight into the problem we will introduce a new decision variable z_j.

$$z_j \leq x_j, z_j \leq y_j, z_j \leq x_j + y_j - 1 \tag{3.18}$$

$$x_j, y_j, z_j \in \{0,1\}, \forall j \in J \tag{3.19}$$

$$nE_1 + \sum_{j=0}^{n} R_{0j} + \sum_{j=0}^{n} (E_1 + T_0 - E_0)x_j + (R_{ij} + T_j - R_{oj} - T_{0j})z_i \leq N_{rg} \tag{3.20}$$

Which can be further simplified as

$$\sum_{j=0}^{n} (E_1 + T_0 - E_0)x_j + (R_{ij} + T_j - R_{oj} - T_{0j})z_i \leq N_{rg} - nE_1 - \sum_{j=0}^{n} R_{0j} \tag{3.21}$$

Representing the problem in this form grants us certain insights The algorithm that we are using is a simple comparison algorithm and the main steps to undergo are as follows, order all the images based on increasing value of R_{ij} +T_j -R_{0j} -T_0, once this is done process all the images on the remote machine such that the energy constraint is satisfied. However one thing to not is that the algorithm itself takes a some energy to run so we should be mindful of the complexity of the algorithm. We will subtract the algorithm energy from the energy budget to get the energy compensated form of this equation 3.21 to get our final equation.

$$\sum_{j=0}^{n} (E_1 + T_0 - E_0)x_j + (R_{ij} + T_j - R_{oj} - T_{0j})z_i \leq N_{rg} - nE_1 - \sum_{j=0}^{n} R_{0j} - E_{alg} \tag{3.22}$$

Chapter 4

Data Description

In this chapter we are going to describe the data-set that we are using for experimentation

4.1 Dataset description

The data used for the experiments comes from the *ImageNet Large Scale Visual Recognition Challenge*, popularly known as the ImageNet dataset. ImageNet is an image database organized according to the WordNet hierarchy (currently only the nouns), in which each node of the hierarchy is depicted by hundreds and thousands of images. The project has been instrumental in advancing computer vision and deep learning research. The data is available for free to researchers for non-commercial use.

ImageNet description: It is a project that was inspired by two important needs in computer vision research. The first was the need to establish a clear North Star problem in computer vision. While the field enjoyed an abundance of important tasks to work on, from stereo vision to image retrieval, from 3D reconstruction to image segmentation, object categorization was recognized to be one of the most fundamental capabilities of both human and machine vision. Hence there was a growing demand for a high-quality object categorization benchmark with clearly established evaluation metrics. Second, there was a critical need for more data to enable more generalizable machine learning methods. Ever since the birth of the digital era and the availability of web-scale data exchanges, researchers in these fields have been working hard to design more and more sophisticated algorithms to index, retrieve, organize and annotate multimedia data. But good research

requires good resources. To tackle this problem at scale (think of your growing personal collection of digital images, or videos, or a commercial web search engine's database), it was critical to provide researchers with a large-scale image database for both training and testing.

ImageNet is an image dataset organized according to the WordNet hierarchy. Each meaningful concept in WordNet, possibly described by multiple words or word phrases, is called a "synonym set" or "synset". There are more than 100,000 synsets in WordNet; the majority of them are nouns (80,000+). ImageNet aims to provide on average 1000 images to illustrate each synset. Images of each concept are quality-controlled and human-annotated. In its completion, we hope ImageNet offers tens of millions of cleanly labeled and sorted images for most of the concepts in the WordNet hierarchy.

4.2 Image subset

Figure 4.1: Sample image 1 for set 1

Figure 4.2: Sample image 2 for set 1

The dataset consists of training and test images. As ImageNet has only provided the training images pre-labeled and categorized while the test images are not labeled or categorized. This led us to use three subsets of only the training data to run the experiments.

This data set is extremely varied with images of varying resolutions (the resolutions don't only vary in length and width of the image but also in picture count and photo quality, with multiple images being noisy and out of focus). The primary reason for picking ImageNet for the dataset in the experiment is its pre-labeled

nature which allows us to test and verify the accuracy of the program without going to the effort of classifying and then manually cross-referencing and confirming a varied test set. So, we have selected three sets from ImageNet which allows us to carry out the experiment to test the robustness of the algorithm with different sets of images. The reason i did not use the test set even if i could handle the labelling by downloading the solved labels was due to the fact that the train set was massive compared to the test set and it allowed me to have a lot of data and study the behaviour of the algorithm with varied data.

The first set is a collection of images of different types of canine animals i.e. dogs of different breeds and canine adjacent animals. The data set consisted of 5000 images varying in resolution from 120×102 to 3264×2448 and thus occupy from 2.94 KB from 1.07 MB.

The second set consists of a variety of images of different species of birds. The second set was a collection of images of different types of birds. The data set consisted of 5200 images varying from 150×150 to 3852×2724 and thus occupy from 2.36 KB 5.35 MB.

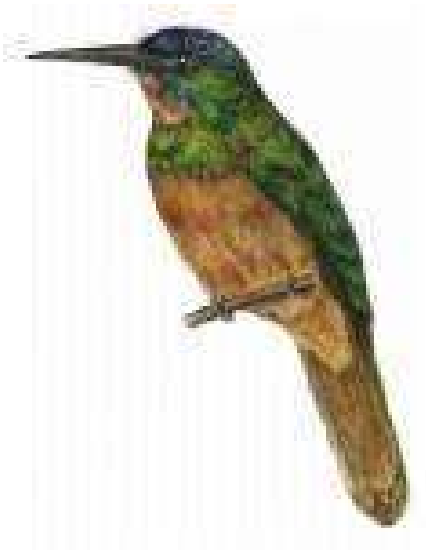

Figure 4.3: Sample image 1 for set 2

Figure 4.4: Sample image 2 for set 2

Figure 4.5: Sample image 1 for set 3

Figure 4.6: Sample image 2 for set 3

The third set consists of trains construction equipment and vehicles. The third set consists of a variety of images of different species of birds. The third set was a collection of images of different types of birds. The data set consisted of 5200 images varying from 100×100 to 1939×2681 and thus occupy from 3.05 KB to 4.46 MB.

Chapter 5

Experimental Setup and Experimentation

To setup our experiment we have to do so at two ends, one being the server and the other being the local machine. Since the two systems are extremely different in nature we have to adopt radically different measurement approaches that work in completely different ways. In this section we shall explore how and why we picked these approaches and the merits and demerits of these approaches, also the specific methods used and how the results are made comparable and to what degree.

5.1 Server side experimentation

This section details the server, its specifications and the server side setup of the experiment

5.1.1 Server Specification

We first need to understand the setup of the experiment, the first of which is the machine that is used as the server, It is a computer with the following specifications:

- Intel i7 6700HQ
- 16 GB DDR4 RAM
- GTX 960M graphics

- Max TDP 120W

5.1.2 Methods to measure the server side energy consumption

We used two methods to measure energy at the server level, the main problem with measuring energy consumption was to find a reliable method to measure energy this would lead us to try to use two different methods both with their own shortcomings and benefits.

Power Top

A computer's electrical power usage can be measured, explained, and reduced using the software tool PowerTOP. In 2007, Intel made it available under the GPLv2 license. It works with processors from Intel, AMD, ARM, and UltraSPARC.

On a machine running the Linux and Solaris operating systems, PowerTOP examines the applications, device drivers, and kernel settings in use and calculates the power usage as a result. The software that consumes excessive amounts of power may be located using this information. Users of laptop computers who want to extend battery life and owners of data centers whose electrical and cooling expenditures are significant may find this to be very helpful.

Of course we are not going to optimise any processes however we are going to use this particular tool to isolate our python script by process ID and monitor its power usage and logging it every 2 seconds to get an idea about the consumption of power for the entire duration of the classification. Here we run into our first issue, there are a lot of processes created while running the classification script and after a little analysis we figure out that the process id of the child processes are nearly identical save the final few digits and the process user is always the same, leveraging these details we can exactly filter out our script and log its wattage, then-after there is an easy equation that is Energy = Power x Time. But since the data points are discrete we have to take the area under the curve for the power vs time graph that we can construct using this collected data. Once we have the area under the curve we can finally say that we have found the energy required to run our script using Power Top.

This however is not a perfect measure in any way, power top itself consumes energy to run along with the script that is required to copy the data from Powertop, the intensive polling done by the script also causes the power consumption to go

	pid	avg_pw	pid_count	energy
0	1936633	0.005850	1	0.0585
1	1939131	0.023800	2	0.4760
2	1939644	54.005125	8	4320.4100
3	1951780	7.340316	45	3303.1424
4	1952128	4.256609	174	7406.5000
...	...	...	...	...
95	1952756	0.354976	59	209.4360
96	1952757	0.400541	56	224.3030
97	1952758	0.329730	47	154.9730
98	1952759	0.396137	52	205.9910
99	1952760	0.312373	59	184.3000

Figure 5.1: Data from Powertop

up. It is difficult to track the process ids of the polling script and distinguish between the polling script and the classifier that is running since both of them were on python. By using the logic of child processes having process ids in the same ball park as the parent process we can only get so far and not guarantee that the polling script did not get mixed up in the energy calculations.

Using An External device to measure the power

An external device which can measure the total energy going to the system at the point of power supply is the best method to track the energy consumption of the system as any other complicated estimate methods like using power top may incur issues as seen above. However as we know any measurement is only as good as the device used to make the measurement. Keeping this in mind we use the *PZEM-022* measurement device to measure the energy consumption.

Figure 5.2: Device Used to Measure Server Energy

In this method we used an energy measurement device mainly the PZEM-022 which has the following specifications:

- Operating Voltage: 80 260V
- Current Rating:100A
- Operating Frequency: 45-65Hz
- Measurement Accuracy: 1.0 grade
- Rated Power: 100A/22000W

- Color: Black
- Wire length:190 mm
- Length: 90 mm
- Width: 50 mm
- Height: 25 mm
- Weight: 90 gm

Since we are operating in the 1-12 A range we have up to a 1 percent accuracy. This is fairly accurate measurement tool but the problem is to measure the classification process and the classification process alone. Step one is to remove the battery from the computer so that no energy is unaccounted for as the battery is no longer a power source. The device is then connected to the power supply to the computer. we run the computer idle for an hour with the same fan speed so as to establish a baseline reading for the power consumption by the computer for running the operating system and background tasks. Once the baseline reading is established we run our script and let the power meter record the energy consumed. Once we have recorded the consumed energy we subtract the baseline to find out the amount of energy consumed by the classification process.

5.1.3 Comparability of energy measurement results

Now using both these methods what we find is there is a nominal difference of 2.6 percent in readings with the power meter measuring slightly more than that of the power top method, therefore the different methods end up giving us results in the same ballpark. This might be due to the problem previously mentioned i.e. the process id for the polling script not being exactly isolated together with the fact that if a process is created and destroyed between polls it will not register at all in the log thereby getting sidelined and not included in the final calculation for energy. This leads us tho the idea that though software might be more focused and tighter in the net that it casts for data, there is a certain Heisenberg-uncertainty thing that goes on in this case where the increase in polling frequency the energy consumption goes up causing a distortion in the energy estimate. For these reasons even though the results are comparable in both methods of measurement, We can have more peace of mind in using the data from the power meter because we are sure of two things, firstly there are no processes that fell through the cracks in between the polling rate and secondly there are no processes that are actually related to the polling script that are being also added to the final measurement.

5.2 Local device side experimentation

To emulate the Local Device we chose a raspberry pi-4 model B with the following CPU specifications:

- Broadcom 2711 Quad-Core Cortex A72 (ARM V8-A) 64-bit SoC Clocked at 1.5GHz
- 8GB RAM
- 6.5W TDP

This raspberry pi will be running Ubuntu desktop which will be running a smaller version of the ImageNet classifier MobilenetV2.

This side of the experiment we were limited by the ability of power top to run properly on the watered down version of linux on the raspberry pi. Therefore we employed the use of UM24C power measuring device with the following specifications

- Voltage range: 4.50 – 24.00V
- Voltage resolution: 0.01V
- Voltage accuracy: ±(0.2%+1 digit)
- Current range: 0.0 – 3.000A
- Current resolution: 0.001A
- Current accuracy: ±(0.8%+3 digit)
- Capacity accumulation range: 0 – 99999mAh
- Energy accumulation range: 0 – 99999mWh
- Load impedance range: 1.5 – 9999.9
- Temperature range: -10°C 100°C / 0°F 200°F
- Temperature error: ±3°C/±6°F
- Voltage graphing range: 4.5 – 24.0V
- Current graphing range: 0.0 – 3.0A
- Screen: 1.44 inch color LCD display

Figure 5.3: Device Used to Measure Raspi Energy

- Quick charge recognition mode: QC2.0/QC3.0
- Refresh rate: 2Hz

We connect the power supply of the raspberry pi through the UM24C device which connects via Bluetooth to another machine and logs the energy ,voltage, power and current data. We connect the Bluetooth on the device to another machine because Bluetooth itself is a power consuming data transmission method and it would impact the readings if we chose to log the data on the raspberry pi.

We then run our raspberry pi for an hour without any load or code executing to estimate a baseline reading for for the power consumption by the computer for running the operating system and background tasks. Once the baseline reading is established we run our script and let the power meter record the energy consumed. Once we have recorded the consumed energy we subtract the baseline to find out the amount of energy consumed by the classification process. we repeat these experiments for the reshaping and transmission processes as well.

5.3 Methodology

In this section we will detail our methods of setting up the experiment and The steps we take to take measurements, we will outline the flow of data and our decision making process.

We are also running our algorithm on the raspberry pi itself and the algorithm also consumes a certain amount of power for its own computation. This would prompt us to add the algorithm energy itself as a factor to consider in the process of energy efficiency of the raspberry pi setup.

Using this insight we can make sure that our algorithm does not cause any extra computation workload on the raspberry pi. This is done by making the algorithm as bare-bones as possible. Apart from the necessary comparisons and look-ups, the algorithm should not perform any unnecessary quality of life operations such as sorting or searching.

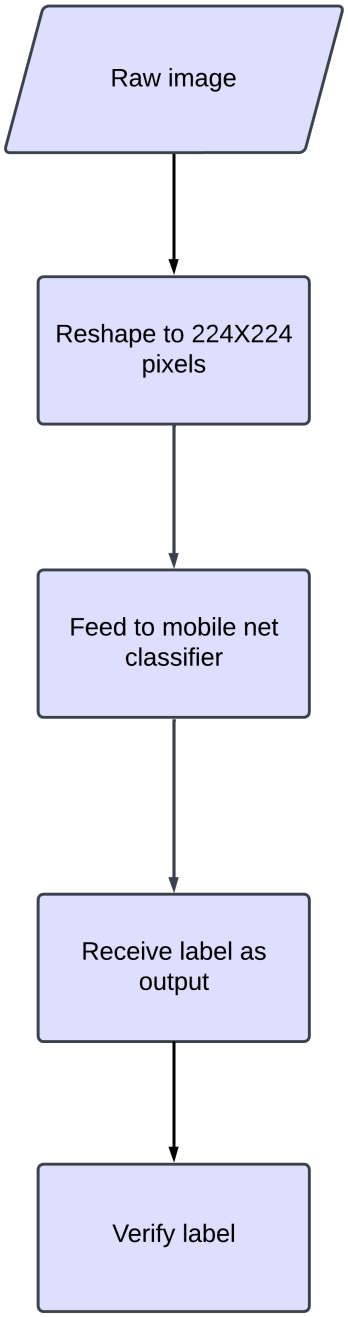

Figure 5.4: General Steps To Classification

The above figure is a flowchart showing how the classification would occur given an absence of any optimisation. The raw image would arrive on the raspberry pi in the real world case that we have imagined it would be. For the purposes of this

experiment however we are using images from the ImageNet dataset as our input images.

Once the raw image arrives it is then reshaped into 224x224 pixels to make it suitable for feeding into the classifier. We will talk about the classifier used and its parameters in the classification models section. This reshaping consumes energy unless the image is already in 224x224. In case of an image any other size the reshape function upscales or downsizes the image as required. More about the reshaping models is covered in the Reshape Function Section.

Once the image is reshaped to the desired size it is then fed to the classification model which takes the image as an input and produces a label that it thinks is the most suitable for the image as output thereby in effect recognising the image. Once the label has been recieved as output we verify the accuracy of the system by matching the label to the actual label of the image and verifying if it is accurate or not.

5.3.1 Classification models

Image classification models are a type of deep learning model designed to recognize and categorize objects or patterns within images. These models are widely used in various applications, including image recognition, content moderation, medical imaging, autonomous vehicles, and more. Here are some notable image classification models:

LeNet-5: Developed by Yann LeCun in the early 1990s, LeNet-5[6] was one of the first convolutional neural networks (CNNs) used for handwritten digit recognition. While it may be considered basic by today's standards, it laid the foundation for modern CNN architectures.

AlexNet: Introduced by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton in 2012, AlexNet[7] significantly advanced the field of computer vision. It featured deep convolutional layers and achieved a substantial reduction in error rates in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), helping to popularize deep learning.

VGGNet: The Visual Geometry Group (VGG) at the University of Oxford proposed VGGNet[8] in 2014. VGGNet is known for its simplicity and uniform architecture, using small 3x3 convolutional filters throughout the network. It was a runner-up in the ILSVRC 2014 competition.

GoogLeNet(Inception): Developed by Google in 2014, GoogLeNet[9] introduced the idea of inception modules, which use multiple filter sizes in parallel and then

concatenate their outputs. This architecture demonstrated strong performance on the ILSVRC dataset while being computationally efficient.

ResNet (Residual Network): Introduced by Kaiming He et al. in 2015, ResNet[10] is famous for its deep residual learning approach. It uses skip connections (shortcuts) to allow the training of very deep networks while avoiding the vanishing gradient problem. ResNet architectures have become the standard for many image classification tasks.

DenseNet: DenseNet[11], proposed by Gao Huang et al. in 2017, takes the skip connections from ResNet to another level. In DenseNet, each layer is connected to every other layer in a feedforward fashion, leading to highly parameter-efficient networks.

EfficientNet: EfficientNet[12], introduced by Mingxing Tan and Quoc V. Le in 2019, focuses on optimizing both model accuracy and efficiency. It uses a compound scaling method to balance the depth, width, and resolution of the network, achieving state-of-the-art results with fewer parameters.

MobileNet: MobileNet [13], developed by Google in 2017, is designed for mobile and embedded vision applications. It utilizes depthwise separable convolutions to reduce computational complexity while maintaining good accuracy, making it suitable for resource-constrained devices.

MobileNet

For our purposes we used MobileNet to classify the images

MobileNet is a family of convolutional neural network (CNN) architectures designed for mobile and embedded vision applications. It was developed by researchers at Google, specifically by Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNet is known for its efficiency, making it suitable for devices with limited computational resources while maintaining reasonable accuracy in image classification tasks. Here are the key features and components of MobileNet:

Depthwise Separable Convolution: MobileNet's primary innovation is the use of depthwise separable convolutional layers. Traditional convolutional layers perform both spatial convolution (using a kernel/filter) and depth convolution (cross-channel convolution) in a single step. In contrast, depthwise separable convolutions split these operations into two separate layers:

Depthwise Convolution: In this step, a separate convolution is applied to each

input channel (depth dimension) independently, using a separate kernel for each channel. This reduces the computational cost significantly, as it requires fewer parameters and computations compared to traditional convolutions.

Pointwise Convolution: After the depthwise convolution, a 1x1 convolution (pointwise convolution) is applied to combine the output channels from the depthwise step. This helps capture cross-channel dependencies and increase the model's expressiveness.

Width Multiplier and Resolution Multiplier: MobileNet introduces two hyperparameters, known as the "width multiplier" and "resolution multiplier," to allow users to scale the model according to their specific requirements:

Width Multiplier (α): This hyperparameter controls the number of channels (filter size) in each layer. A smaller value reduces the number of channels, leading to a smaller model with fewer parameters and lower computational requirements. A larger α value increases the model's capacity at the cost of more computation.

Resolution Multiplier (ρ): This hyperparameter scales down the input image resolution. Reducing the resolution lowers the computational demands further. Typically, MobileNet models are designed to be efficient across a range of resolutions.

Architectural Variants: There are several architectural variants of MobileNet, including MobileNetV1, MobileNetV2, and MobileNetV3. Each variant aims to improve upon the previous one in terms of both efficiency and accuracy.

MobileNetV1: The original MobileNet[13] architecture, introduced in 2017, laid the foundation for depthwise separable convolutions. It achieved good efficiency but with some limitations in terms of accuracy.

MobileNetV2: Released in 2018, MobileNetV2[14] improved upon the original by introducing inverted residual blocks with linear bottlenecks. It achieved better accuracy while still being efficient.

MobileNetV3: MobileNetV3[15], introduced in 2019, further improved accuracy and efficiency through the use of a combination of depthwise and pointwise convolutions and other architectural innovations.

Applications: MobileNet models are particularly well-suited for tasks like image classification, object detection, and semantic segmentation on resource-constrained devices, such as mobile phones, embedded systems, and IoT devices. They strike a balance between model size, inference speed, and accuracy.

MobileNet has become a popular choice for various real-time computer vision applications, thanks to its efficient design and adaptability. The choice of which

MobileNet variant to use (V1, V2, or V3) depends on the specific requirements of the application and available computational resources.

This is the perfect model to run on small computers and edge devices with limited computing potential. This is the model we will be using for the classification process on the raspberry pi which is our remote device.

We chose Mobilenet V2 over V3 for our remote device for two basic reasons,

- It is easy to set up MobileNet V2 on a raspberry pi since it is an earlier version. Most of the kinks in its implementation have been ironed out for remote devices.
- The MobileNet V3 model is marginally more accurate as its main gains in performance were about latency. We already have a handle on this because we can modify the alpha value of MobileNet V2 but this work does not really pertain to latency so those gains are not of much value to us.

We chose MobileNet V3 for the server because its more accurate than MobileNet V2 and its easier to run on the specifications of our server.

5.3.2 The Algorithm

This section deals with the algorithm for the decision making process of the offloading of task.

The Algorithm itself is very simple can easily be understood by the means of these two flow charts Fig 5.5 and Fig 5.6

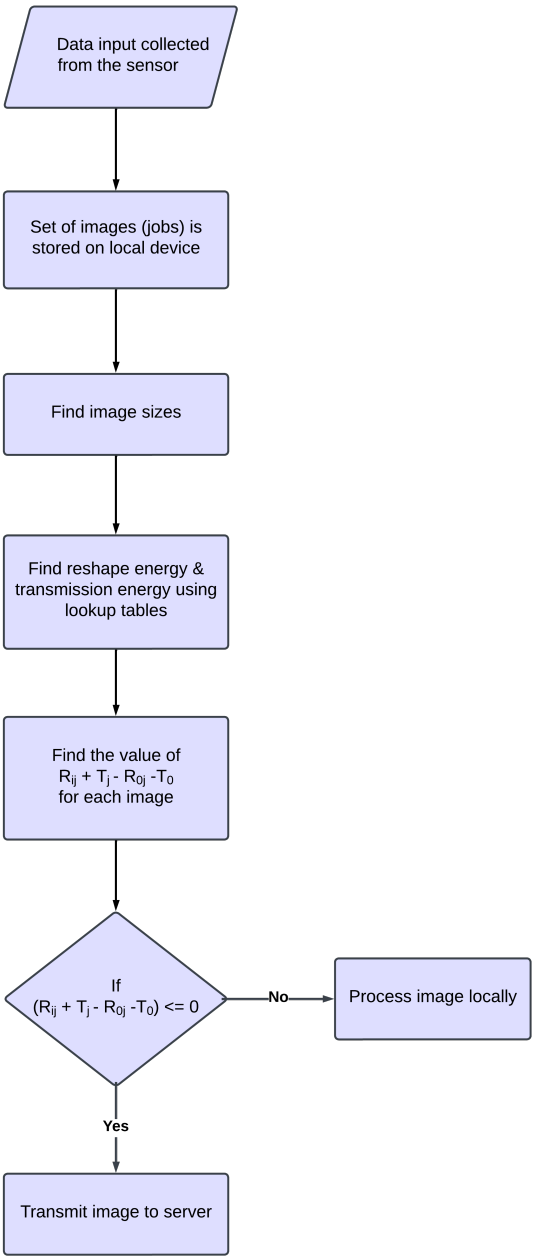

Figure 5.5: Algorithm flowchart 1.

Local Machine Side

From this figure we can follow all the step of the algorithm. At first the data is collected from the sensors as input. This data is then stored on board the edge device's memory.

Then we find the size of each image. This is an important step as the size of the image is the best identifier of the energy that will potentially be used to process it.

Next we use the lookup tables that we have on board the edge device to figure out the different potential estimates of different energy values. The values that are being looked at are from three separate lookup tables. The first lookup table pertains to the reshaping energy of the server, the graph for which is found in Figure 5.8. We find the closest value we have on the lookup table then we average for our required value that best describes our case.

our next step is to find the value of the expression R_{ij} + T_j - R_{0j}-T_{0j} which is pretty straightforward as we have all the values for each of the terms from the lookup table.

We find that sometimes this expression yields a negative value. This is the golden window of offloading the data. Whenever this condition is met the image is offloaded to the server.

This offload however take place at the cost of transmission energy. The transmission energy intuitively scales with image size. Therefore it is extremely costly in terms of energy to offload images over a certain size. The image offloading was tested over WiFi so all the energy consideration and measurements are specific to the 2.4 GHz 802.11ac Wi-Fi technology. The graph for the lookup table for the same can be found in Figure 5.8

Once the image has been offloaded the server takes over and processes the image according to the algorithm flowchart 2 in fig 5.6.

Server Side

The server receives the image. The first job it has to do is then to find out if the image that has been received is already reshaped or not. If it is reshaped then the server proceeds to use the image as it is, if it is not in the 224x224 required size the server runs the reshape function on the image to reshape it to the required specification.

Once the image is in the correct dimension, the server then proceeds to classify

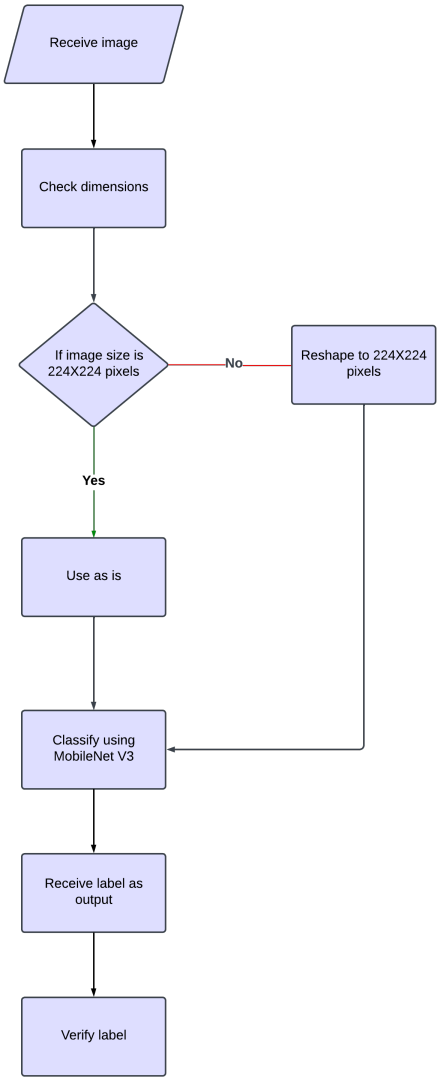

Figure 5.6: Algorithm flowchart 2.

the image using the Mobilenet V3 image classifier.

After classifying the image we receive a label as an output, this label is then matched with the original label for the image and the accuracy is established.

Why no Reshape and Transmit

The reason we don't really have reshape and offload as a viable option here is because the server reshape function is effectively cheaper in energy cost than the local device due to difference is computing power. The local device has less computational resources and almost no graphical processing capabilities that can make the reshaping easier. Since reshaping is mostly matrix multiplication which is in itself a bunch of multiply and accumulate expressions the graphical processors onboard the server are far more efficient at reshaping than the CPU of the local device. This causes the decision to reshape and transmit costlier than transmit and reshape for our sweet spot. If we could shift the sweet spot we could unlock more energy savings which brings us to our second reason. The second reason is that in the problem statement we value the energy at the local device and server equally which is to say that both the energy at the server and the energy at the local device are equally valuable in the equation. There is a weighted version of this problem that has been suggested to be worked on in the future and that version has the option of assigning different weights to the local machine energy and the server energy and even possibly dynamically changing the weights as the local device energy depletes. since the server reshape is less energy consuming than local reshape it is always the clearer choice in the case of transmission energy being comparable to the reshaping energy.

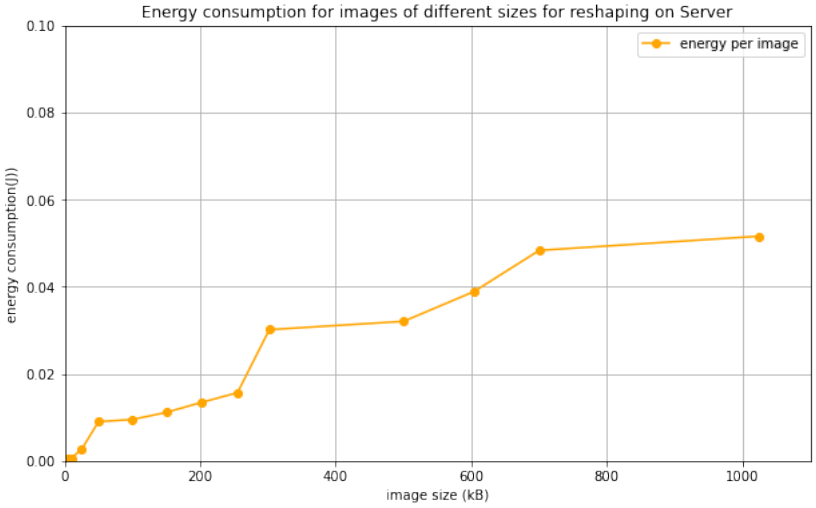

Figure 5.7: Reshape Energy Lookup Table Graph

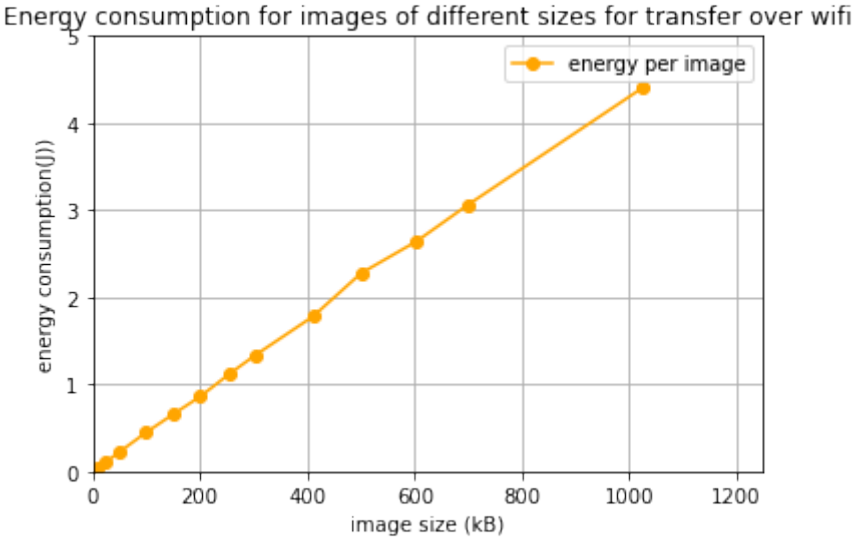

Figure 5.8: Transmit Energy Lookup Table Graph

Chapter 6

Results and Observations

We perform all the experiments to measure a baseline and calculate the minimum energy budget required to do all jobs in a particular dataset. The first quantity we have to measure is the Average power consumption when the raspberry pi is idle to establish a baseline of power consumption. We ran the test for an hour using the UM24C which logs voltage and current data as the pi runs with a frequency of once every second. The power is calculated by multiplying the voltage and current at each reading and then averaging it over the time the setup was running the classification task. The Average power at idle is found to be 2.856935W. This measurement is set as the baseline for all further measurements.

Here are the results for the energy budget of the raspberry Pi for all three sets of data.

6.1 Data-Set 1 Energy Budget and Improvements

This section details the calculation of the energy budget for the first dataset.

Then by using a similar method to how we calculated the baseline power consumption we found the average power while classifying images which was found to be 4.205253W and this power was averaged over 1225 seconds. We took the number of seconds it took to classify the entire data-set and multiply it with the power figure to find the associated energy consumption.

$$P = W/T \tag{6.1}$$

where work done is equal to energy expended therefore.

$$W = P \times T \tag{6.2}$$

using 6.2 we get **1651.69 Joules** of energy consumed.

1651.69 Joules is our energy budget for dataset 1.

After this We implemented our algorithm and re-run the classification to find out the energy consumption while the algorithm was running. We found the average power while classifying images which was found to be 4.177073 W and this power was averaged over 1225 seconds. We took the number of seconds it took to classify the entire data-set and multiply it with the power figure to find the associated energy consumption.

Using equation 6.2 we get **1,619.81 Joules**

We also find that during the course of this that 170 images were offloaded to the server which had an average accuracy of 75 % for this set and an average gain in accuracy of .42% due to this offloading. Initial accuracy for this set for the remote device was 62.8% and after the algorithm it was 63.22%

This leads us to find that there was an improvement of 1.93 percent in energy consumption and we are under budget.

Energy consumed by the server while the classification ran set 1= 887.34

6.2 Data-Set 2 Energy Budget and Improvements

This section details the calculation of the energy budget for the second data-set.

By using a similar method to how we calculated the baseline power consumption we found the average power while classifying images which was found to be 4.280284 W and this power was averaged over 1275 seconds. We took the number of seconds it took to classify the entire data-set and multiply it with the power figure to find the associated energy consumption.

using 6.2 we get **1814.77 Joules** of energy consumed.

1814.77 Joules is our energy budget for data-set 2

After this We implemented our algorithm and re-run the classification to find out the energy consumption while the algorithm was running. We found the average power while classifying images which was found to be 4.262346 W and this power

was averaged over 1275 seconds. We took the number of seconds it took to classify the entire data-set and multiply it with the power figure to find the associated energy consumption.

using 6.2 we get **1,791.90 Joules**

We also find that during the course of this that 144 images were offloaded to the server which had an average accuracy of 76 % for this set and an average gain in accuracy of .36% due to this offloading.Initial accuracy for this set for the remote device was 64.1% and after the algorithm it was 64.47%

This leads us to find that there was an improvement of 1.26 percent in energy consumption and we are under budget.

Energy consumed by the server while the classification ran set 2= 693.46

6.3 Data-Set 3 Energy Budget and Improvements

This section details the calculation of the energy budget for the Third data-set.

By using a similar method to how we calculated the baseline power consumption we found the average power while classifying images which was found to be 4.239627 W and this power was averaged over 1263 seconds. We took the number of seconds it took to classify the entire data-set and multiply it with the power figure to find the associated energy consumption.

using 6.2 we get **1746.34 Joules** of energy consumed.

1746.34 Joules is our energy budget for data-set 1.

After this We implemented our algorithm and re-run the classification to find out the energy consumption while the algorithm was running. We found the average power while classifying images which was found to be 4.177073 W and this power was averaged over 1225 seconds. We took the number of seconds it took to classify the entire data-set and multiply it with the power figure to find the associated energy consumption.

using 6.2 we get **1730.78 Joules**

We also find that during the course of this that 130 images were offloaded to the server which had an average accuracy of 75 % for this set and an average gain in accuracy of .22% due to this offloading.Initial accuracy for the remote device for this set was 62% and after the algorithm it was 62.22%

This leads us to find that there was an improvement of 0.891 percent in energy consumption and we are under budget.

Energy consumed by the server while the classification ran set 3= 590.85

6.3.1 Compiled Results

With all of this data collected we can see improvements in the energy consumption and accuracy compared to processing completely on the edge device. We can also understand that the results vary from data-set to data-set. This has a correlation with size of the images. Due to the nature of the reshaping function and its higher cost on the edge device due to low computation capabilities, Therefore if the images are easier to transmit than to reshape it makes sense to send the images to the server this causes us to play around a sweet spot in the reshape function that occurs when the reshape function switches from up-scaling to down-scaling. Every image that fits this sweet spot can be offloaded without sacrificing any performance. Therefore the results are limited by number of such images. If we come across a data set with more of these sweet spot images we can expect anywhere from 5-10% boost in energy savings. Here is a graph showing the energy consumption of

Results and Observations

METRIC	WITHOUT ALGORITHM (DATA SET 1)	WITHOUT ALGORITHM (DATA SET 2)	WITHOUT ALGORITHM (DATA SET 3)	WITH ALGORITHM (DATA SET 1)	WITH ALGORITHM (DATA SET 2)	WITH ALGORITHM (DATA SET 3)
TOP 1 ACCURACY (%)	62.8	64.1	62	63.22	64.47	62.2
IDLE AVERAGE POWER CONSUMPTION (WATTS)	2.856935			2.856935		
- WHILE CLASSIFYING IMAGES	4.205253	4.280284	4.239627	4.177073	4.262346	4.227307
IMAGES OFFLOADED	0	0	0	170	144	130
ENERGY CONSUMPTION (JOULES)	1651.69	1814.77	1746.34	1,619.81	1,791.90	1,730.78
% IMPROVEMENT WITH ALGORITHM & OFFLOADING	-			1.93%	1.2602%	0.891%

Figure 6.1: Results and Observations

different data sets and comparing it to the case with the algorithm running.

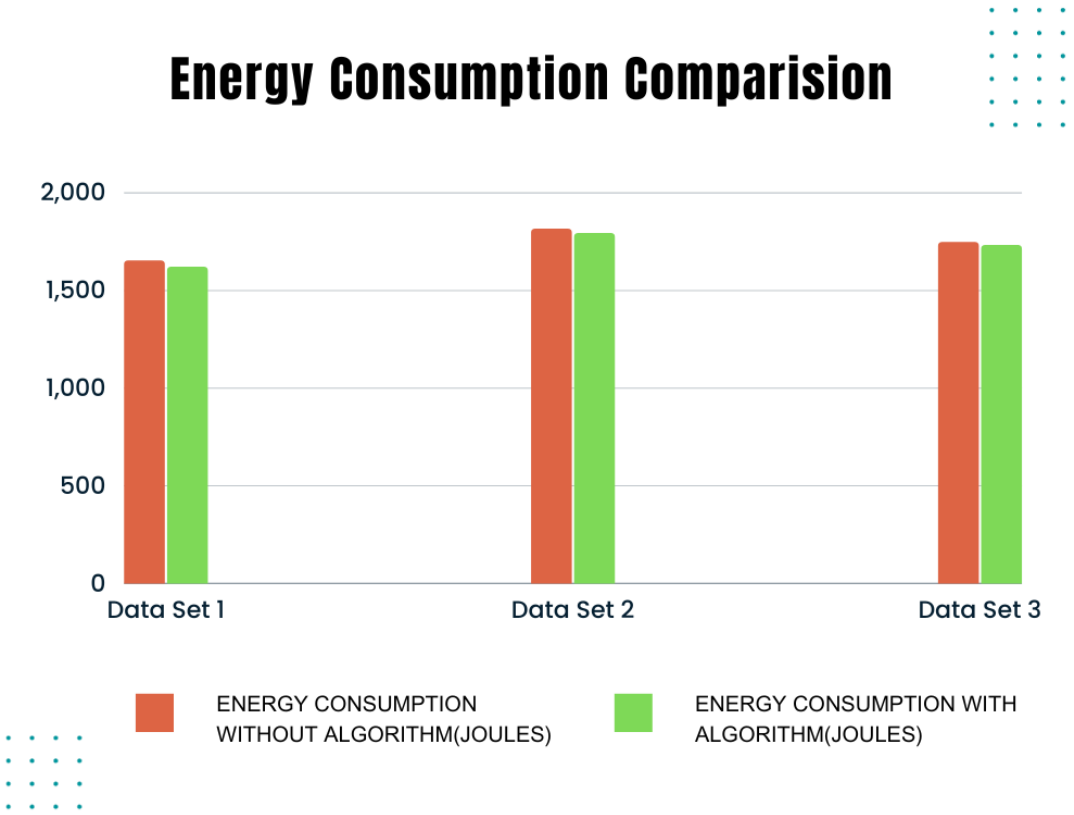

Figure 6.2: Graph for energy consumption comparison

Chapter 7

Conclusion

Based on the results obtained from our experiments, we can draw the following conclusions:

1. Energy Efficiency with Offloading: When we implemented the algorithm on the Raspberry Pi with the capability to offload certain tasks, we observed an improvement in energy efficiency compared to without running the algorithm offloading. This improvement was consistent across all three Pi sets, albeit to varying degrees.

2. Energy Savings Vary: The degree of energy savings depended on the specific Pi set and the nature of the tasks being offloaded. In our experiments, we achieved energy savings ranging from 0.891% to 1.93% when compared to running the algorithm without offloading. This demonstrates the potential benefits of task offloading in energy-constrained environments.

3. Optimizing Offloading Strategies: To further enhance energy efficiency, future work could focus on optimizing the offloading strategies, One such optimisation might be general forms of the equations used here. Generalising such a concept for all networks could work wonders because energy savings add up over time.

4. Future Prospects: There is a version of this problem which deals with weights on energies. The server energy and the edge device energy are valued differently, possibly even dynamically changing weights depending upon the situation. An algorithm that can work for such a problem might be very useful in every device that has a limited power source.

In summary, our research demonstrates the potential benefits of task offloading for improving energy efficiency in Edge devices. By optimizing offloading strategies

and tailoring them to specific applications, we can save a lot of energy. This is of great importance in resource-constrained environments and for achieving sustainable computing solutions and of-course better for the world one joule at a time.

Bibliography

[1] D Suleiman, M Ibrahim, and I Hamarash. «Dynamic voltage frequency scaling (DVFS) for microprocessors power and energy reduction». In: *4th International Conference on Electrical and Electronics Engineering*. Vol. 12. 2005 (cit. on p. 4).

[2] Ke Zhang, Yongxu Zhu, Supeng Leng, Yejun He, Sabita Maharjan, and Yan Zhang. «Deep learning empowered task offloading for mobile edge computing in urban informatics». In: *IEEE Internet of Things Journal* 6.5 (2019), pp. 7635–7647 (cit. on p. 5).

[3] Feng Wei, Sixuan Chen, and Weixia Zou. «A greedy algorithm for task offloading in mobile edge computing system». In: *China Communications* 15.11 (2018), pp. 149–157 (cit. on p. 5).

[4] Sirine Bouhoula, Marios Avgeris, Aris Leivadeas, and Ioannis Lambadaris. «Computational offloading for the industrial internet of things: A performance analysis». In: *2022 IEEE International Mediterranean Conference on Communications and Networking (MeditCom)*. IEEE. 2022, pp. 1–6 (cit. on p. 6).

[5] Ying Chen, Ning Zhang, Yongchao Zhang, Xin Chen, Wen Wu, and Xuemin Shen. «Energy efficient dynamic offloading in mobile edge computing for internet of things». In: *IEEE Transactions on Cloud Computing* 9.3 (2019), pp. 1050–1060 (cit. on p. 6).

[6] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. «Gradient-based learning applied to document recognition». In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324 (cit. on p. 32).

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. «ImageNet Classification with Deep Convolutional Neural Networks». In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger. Vol. 25. Curran Associates, Inc., 2012. URL:

https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf (cit. on p. 32).

[8] Karen Simonyan and Andrew Zisserman. «Very deep convolutional networks for large-scale image recognition». In: *arXiv preprint arXiv:1409.1556* (2014) (cit. on p. 32).

[9] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. «Going deeper with convolutions». In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9 (cit. on p. 32).

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. «Deep residual learning for image recognition». In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778 (cit. on p. 33).

[11] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. «Densely connected convolutional networks». In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 4700–4708 (cit. on p. 33).

[12] Mingxing Tan and Quoc Le. «Efficientnet: Rethinking model scaling for convolutional neural networks». In: *International conference on machine learning*. PMLR. 2019, pp. 6105–6114 (cit. on p. 33).

[13] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. «Mobilenets: Efficient convolutional neural networks for mobile vision applications». In: *arXiv preprint arXiv:1704.04861* (2017) (cit. on pp. 33, 34).

[14] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. «Mobilenetv2: Inverted residuals and linear bottlenecks». In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 4510–4520 (cit. on p. 34).

[15] Andrew Howard et al. «Searching for mobilenetv3». In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 1314–1324 (cit. on p. 34).