Power Consumption Analysis during Image Classification for IoT Nodes

Anand Krishnan Prakash
Energy Science Technology & Policy
anandkrp@andrew.cmu.edu

Nandini Ramakrishnan
Electrical and Computer Engineering
nramakri@andrew.cmu.edu

ABSTRACT
Devices equipped with network connectivity, are becoming increasingly pervasive in day-to-day life (for example. smart homes). The data logged in these devices is used for analytical purposes, like understanding human behavior and habits for providing better services like a smart thermostat. Such networked devices are colloquially referred to as Internet of Things (IoT). However, while devices that support IoT can be deployed to collect data, the on-board processing power may not be sufficient to produce results accurately and within the required time frame. The work in this paper does a performance, power and temperature analysis of performing heavy computation on a IoT device versus performing this computation on a server. The results of this analysis is also presented in this paper. One of the main observed insights at this point in the project is that the time taken, power consumption varies with the frequency scaling policy chosen and that server execution with the most conservative voltage scaling policy on the board takes the least time, but this communication consumes a lot of energy.

Keywords
Internet of Things; Raspberry Pi; Image Classification; Neural Network; Power Meter; Frequency Scaling

1.INTRODUCTION
There is a tremendous growth of cyber-physical systems and ubiquitous sensing. Different kinds of sensors are being deployed everywhere to collect data. Sensor nodes are becoming more powerful and are able to be connected to the internet as well – entering the paradigm of Internet of Things (IoT). IoT-capable systems are extensively being deployed. They can collect data, perform analysis and also take the necessary actions based on the analysis. This becomes more important in the case of real time tasks where collecting data, transmitting it to server and getting the necessary response back takes more time. For example, the Thermal Comfort Band Maintenance (TCBM) algorithm [2] – that maintains a temperature band in room with less number of air conditioners by cycling between the air conditioners - uses a Raspberry Pi to collect data from temperature sensors and then analyses it on-board to make a decision when and which air conditioners to turn on. But this increases the processor workload and hence more power is consumed.

With the on-set of IoT, sensor nodes are becoming more powerful and are able to do heavy processing. But this increases the power consumption and hence leads to a decrease in operating life as most of these nodes are battery operated. Hence the goal of this paper is to evaluate computation versus communication in IoT nodes. We achieve this by quantifying power consumption and time taken for executing computationally intensive tasks in different scenarios - on the IoT node, or on a server talking to the node.

In this project, both the scenarios will be implemented and the results – power consumption, time taken and temperature during both on-board and on-server cases will be analyzed. This will contribute to having a better understanding of energy aware Internet-of-Things design. Section 2 describes the problem statement, Section 3 talks about the experimental setup and the approach that has been taken for this comparative study. Section 4 provides the results and outputs that were observed during running tests until Milestone 1, Section 5 contains the individual contributions for this project and the paper concludes with Section 6 that talks about future work and milestones yet to be reached.

2.PROBLEM STATEMENT
For a board that supports Internet of Things, taking into consideration the power consumption of the board, we will perform a computation versus communication analysis for executing image classification algorithms. The main requirement for a development board to support the internet of things is the availability of network connectivity. Results will include the power, performance and latency analysis when the image classification algorithm is executed on the board and when the image is sent to a server that executes it and sends the result back to the board.

2.1.Image Classification
Image classification is a processor intensive task and can be used to maximize the workload on the board's processor. Image classification on embedded systems and IoT nodes also have a wide range of applications. One of them is occupant identification in a building and using the occupant information to change the environmental setting of the room – such as the room temperature (HVAC [3]) or lighting to match the user's preference.

This paper talks about digit classification on the MNIST database [4], which contains handwritten digits. It has

60000 images in the training data set and 10000 testing images. Digit classification is carried out using neural networks. The LeNet network [10] is implemented using the Caffe framework [11].

Figure 1: MNIST database

3.EXPERIMENTAL SETUP

3.1.Requirements

As per our problem statement, we require a micro controller unit which is compact enough to serve as an IoT node, yet capable of performing compute-heavy exercises such as image classification. It should also have networking capabilities such as Bluetooth or Wi-Fi. in order to communicate with a server. We decided to use the Raspberry Pi 3, a powerful micro controller unit with a quad-core ARM processor, which comes with a built-in Wi-Fi module. The Raspberry Pi also has enough RAM to be able to process 10,000 test images from the MNIST database.

We decided to choose the UbuntuMate over Raspbian operating system for the Raspberry Pi because it is an Ubuntu distribution – hence it would be able to provide lot more support. This gives us access to more Linux tools than Raspbian. As Ubuntu is also built on Debian, it gives all the functionalities that Raspbian could provide. This also provides us with a terminal for running and executing the code and other scripts locally on the Raspberry Pi.

Table 1: System overview

Hardware	Software
Raspberry Pi 3	Caffe
Power Meter	vcgencmd
	Python sockets

3.2.Approach

The problem requires us to compare two situations based on where the process of image classification takes place: on-board or remote.

To perform digit classification easily and with high accuracy, we chose the Caffe deep-leaning framework on which we ran an implementation of LeNet - a convolutional neural network which is known to perform well with the digit classification task. Caffe and its dependencies were installed on the Raspberry Pi.

As a pre-processing step, before beginning any data collection, the neural network model to perform digit classification is pre-trained on a server. The model which is generated is then copied onto the Raspberry Pi. The on-board and remote classification tasks which are subsequently described only look at the testing phase, as the training and verification phases are completed during pre-processing. The results are obtained in the form of accuracy and cross-entropy loss. For our trained model, the accuracy and loss were consistently 98.6 % and 0.043 respectively.

3.2.1 Onboard classification

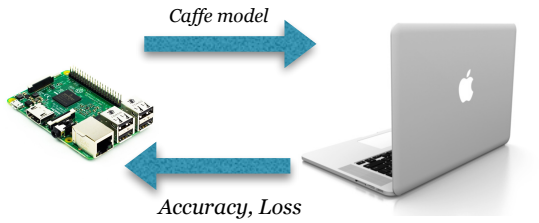
In the on-board implementation, image classification occurs on the Raspberry Pi itself.

3.2.2 Remote classification

In the current server implementation, the Raspberry Pi will send the training data and the testing data of the MNIST database to a server over Wi-Fi using sockets. Python sockets have been used to implement this data transmission part. The server will run the same image classification algorithm, which was run during the onboard implementation, obtain a result and transmit it back to the Raspberry Pi. The power measured in this case will include the power consumed from when the training set was sent, to when the result was received from the server. The time taken is the same duration – from sending the training set to receiving the image, which we obtain by time-stamping throughout the classification process.

We have used a MacBook Pro laptop with high configurations (Core i7, 16GB RAM) as the remote server location for image classification.

Figure 2: Remote classification data flow



3.2.3 Frequency Scaling

We also wanted to analyse the effects of frequency scaling on the power consumption during the on-board and remote classification tasks. By slowing down the frequency, we reduce power, but on the flip-side, the time taken to complete the task increases. To understand the effects of

different frequencies on the results, we modify the clock rate by modifying the Raspberry Pi's system files, which are open to modifications provided a user has root access.

The Raspberry Pi 3 has 2 main operating clock frequencies: 600MHz and 1200MHz. It also comes with five different frequency scaling governing policies for adjusting the clock frequency of the Pi at a given time [8]:

- ondemand (default): runs at 600MHz until CPU usage is 95% (can be configured) and then runs at 1200MHz
- powersave: always runs at 600MHz
- performance: always runs at 1200MHz
- conservative: gradual scaling of clock frequency from 600MHz to 1200MHz based on CPU usage
- userspace: according to user's settings

Our analysis includes results with ondemand, powersave and performance policies. The policy can be changed by overwriting the content of the scaling_governor file in any cpu (/sys/devices/system/cpu/cpu[0-3]/scaling_governor).

3.2.Methods of Data Collection and Analysis

3.2.1.Power Meter

Power meters, or electricity usage monitors, are devices which can be plugged into a wall socket between the socket and the appliance. As shown in Figure It contains an LCD display which gives you real-time accurate information on the power being consumed by your appliance in the form of volts, amperes and watts. The power meter is plugged in at all times to get this information from the Raspberry Pi, while it is running onboard and remote classification. We have used the Kill-A-Watt [6] power meter for this project. Due to lack of network connectivity on the meter, the data had to be manually recorded.

Figure 3: Kill-A-Watt Power Meter

3.2.4.vcgencmd [9]

This is a linux tool that provides us with different system information like current core voltage, current core frequency, LED status on the Raspberry Pi amongst other information. We use this command to retrieve core temperature as well (*vcgencmd measure_temp*). We have written a shell script that runs in the background to collect the core temperature and write to a file every 5 seconds.

4.RESULTS

We tested six different scenarios, with various combinations of onboard and remote server execution of our classifier, with powersave, performance and on demand frequency adjustment policies.

4.1.Power consumption

The tables below summarize our results. We are presenting average power, execution time and our figures of merit - power delay product and energy delay product.

Table 2: Average power consumption during execution

Frequency policy	Approach	Average power (W)
On-demand	Remote	1.7
On-demand	On-board	4.74
Power-save	Remote	1.74
Power-save	On-board	2.76
Performance	Remote	2
Performance	On-board	4.78

Table 3: Power-delay product

Frequency policy	Approach	Power-Delay Product (W-s)
On-demand	Remote	156.2
On-demand	On-board	94.86
Power-save	Remote	116.9
Power-save	On-board	121.24
Performance	Remote	103.5
Performance	On-board	95.6

The results for average power follow our expectations, where average power is significantly reduced during the remote execution. The maximum is obtained during on-board performance mode, as expected. The average power table clearly shows the effects of frequency scaling, where on-demand and power save modes are comparable and on-demand gives a slight edge due to higher frequency performance at high workload.

Table 4: Energy-delay product

Frequency policy	Approach	Energy-Delay Product ($W\text{-}s^2$)
On-demand	Remote	14214.2
On-demand	On-board	1897.2
Power-save	Remote	7715.4
Power-save	On-board	5334.56
Performance	Remote	5485.5
Performance	On-board	191.2

Table 5: Execution Time

Frequency policy	Approach	Time (s)
On-demand	Remote	91
On-demand	On-board	20
Power-save	Remote	66
Power-save	On-board	44
Performance	Remote	53
Performance	On-board	20

However, once we start accounting for the total execution time, we find that the results start to get flipped and remote-execution of the task actually returns worse figures of-merit than on-board processing.

The effects of including execution time get further exacerbated when taking into account energy-delay product.

4.1.2.Effects of frequency and execution time

Remote execution took tens of seconds longer than on-board execution. The remote execution time is a large factor in why it consumes a lot more power.

4.2.Temperature

As expected, the temperature saw a spike during execution of the image classification model. Figure 4 plots the different temperature profiles during the execution period.

It can be seen that onboard execution causes a higher temperature than remote execution. Executing onboard with performance frequency policy shows reduction in temperature. This could be due to very high starting temperature due to programs which were running previously.

4.3.Sources of error

4.3.1 Power meter results

Due to the manual result-taking from the power meter, power and temperature graphs for the different scenarios are approximate, and may come within a few seconds of error.

4.3.2 Temperature skewing with time

The temperature also depends on the amount of time the Raspberry Pi is on for. The longer the testing process took, the likelier it was for scenarios tested later to exhibit a skewed increase in temperature (in the order of 0.1 degrees Celsius). A methodology to normalize this over time will contribute to more accurate results in the future.

4.3.3 Network connectivity

The remote classification values are highly dependent on the speed and congestion of the network, and will vary between networks.

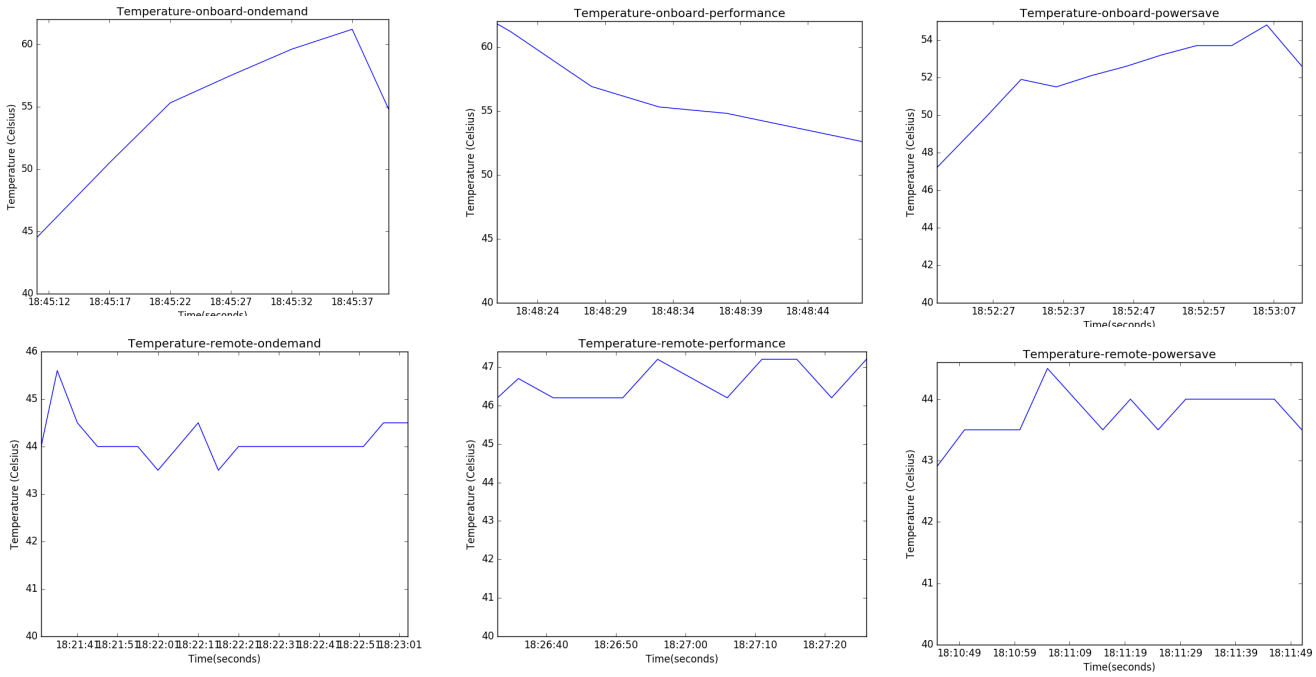
4.3.4 Power consumed by server

The power consumed by the server was not counted in the total consumption for remote processing. The values for remote processing are more optimistic without this added value.

5.CONTRIBUTIONS

Most of the work was done together – set up, data collection, data analysis, setting up Caffe, creating

Figure 4: Temperature Profile for different scenarios



presentations and writing the final report. Nandini wrote the bit classification script to obtain initial Support Vector Machine results. Anand wrote the file and result transfer part from Raspberry Pi to the server and back.

6.CONCLUSION

As expected, temperature rose during processing of image classification, and the value by which it increased remained fairly constant throughout the different cases.

On-board digit classification outperforms remote classification for all frequency modes. The direct effects of frequency policy on energy savings are inconclusive. The power consumption depended more on the execution time of the task, than on any frequency policy.

From our experimental setup, we conclude that the Raspberry Pi is very well suited for computing tasks itself, rather than communicating its tasks to a server.

7.MILESTONES

For Milestone 2, we had done our preliminary analysis of temperature and power consumption while doing image classification with support vector machine as the learning algorithm. As we hadn't used Caffe or any other framework, the algorithm wasn't parallelized. The Raspberry Pi also ran our of memory when we were training and it we trained it for only 20000 images instead of 60000 images. Also, the support vector machine was performing binary classification in Python, where we classified numbers as 0 or not 0. We have since replaced this by running LeNet on Caffe for the same task, and recalculated our previous results.

8.CONTRIBUTIONS

Majority of the work was done together - beginning from the set up, data collection, data analysis. Nandini set up Caffe and wrote the bit classification using SVM for MNIST dataset. Anand developed the script for sending images to the server from the Raspberry Pi and getting the response back. The poster and this report was also compiled together.

9.WEBSITE

A summary of the report is uploaded to this website: http://www.andrew.cmu.edu/user/anandkrp/

10.REFERENCES

1. G. Karmakar, A. Kabra and K. Ramamritham, "Coordinated scheduling of thermostatically controlled real-time systems under peak power constraint," *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2013 IEEE 19th*, Philadelphia, PA, 2013, pp. 33-42.doi: 10.1109/RTAS.2013.6531077, URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6531077&isnumber=6531071
2. M. Feldmeier and J. Paradiso. Personalised HVAC control system. In Internet of Things (IOT), 2010, pages 1 –8, 29 2010-dec. 1 2010.
3. LeCun, Yann, Corinna Cortes, and Christopher JC Burges. "The MNIST database." http://yann.lecun.com/exdb/mnist (1998).
4. Winder, S. A. (2015, July 14). *Training neural nets on MNIST digits.* Retrieved from Simon Winder: http://simonwinder.com/2015/07/training-neural-nets-on-mnist-digits/
5. Kill A Watt: http://www.p3international.com/products/p4400.html
6. scaling_governor: https://wiki.archlinux.org/index.php/CPU_frequency_scaling#Scaling_governors
7. vcgencmd: http://www.elinux.org/RPI_vcgencmd_usage
8. Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, november 1998.
9. Jia, Yangqing and Shelhamer, Evan and Donahue, Jeff and Karayev, Sergey and Long, Jonathan and Girshick, Ross and Guadarrama, Sergio and Darrell, Trevor. "Caffe: Convolutional Architecture for Fast Feature Embedding" (2014) Retrieved from arXiv preprint arXiv:1408.5093