MEng Individual Project

Imperial College London

Department of Computing

# Energy optimisation of Cascading Neural-Network classifiers

*Author:*
Vinamra Agrawal

*Supervisor:*
Dr Anandha Gopalan

*Second Marker:*
Dr Naranker Dulay

June 17, 2019

## Abstract

As Neural Networks become more capable of solving ever-increasing challenging problems, they are bound to be more energy-consuming in the future. Therefore it is essential to develop techniques to conserve energy usage of these networks to balance their environmental impact. In this project, we study multiple techniques to reduce the energy consumption while evaluation of Neural Networks without significantly reducing the accuracy of the network. We found Cascading Neural Networks to be particularly interesting as they offer the ability to reduce the energy consumption with no significant difference in the accuracy of the predictions and without using any specialised hardware.

To improve these Cascading Neural Networks, we show that using semantic data such as Colour, Edges, Texture and Corners from the input images and systematic techniques such as PCA and LDA reduce the dimensions of the input space. This reduced complexity in input allowed us to create simpler classifiers which are more energy-efficient. We further generate an algorithm to arrange these simple classifiers such that they can be energy-efficient without taking a toll on the accuracy of the overall system. We were able to achieve a 13% reduction in energy consumption over the Scalable effort classifiers algorithm and 35% reduction when compared to Keras CNN for Cifar10.

Finally, we also reduced the energy consumption of the established neural network; which is used as the last stage in the cascading technique. We used Bayesian optimisation with adjustable parameters and minimal assumptions to search for the best model under the given energy constraint. We show that by employing the above technique, significant energy saving (29% and 34% for MNIST and Cifar10 datasets) can be achieved which would ultimately contribute to making energy consumption in the Neural Networks more sustainable.

**Acknowledgements**

I want to thank Dr Anandha Gopalan for his guidance and enthusiasm throughout the project. His support and dedication towards the project were un-parallel even during his vacation time. I am also grateful towards Mr Riccardo Moriconi and Dr Naranker Dulay for providing me insights and ideas for topics to explore for this project. I would also like to express my gratitude to my family for their moral support and encouragements.

# Contents

# List of Figures

Chapter 1

# Introduction

## 1.1 Motivation

Machine learning is an evolving branch of computational algorithms that are designed to emulate human intelligence by learning from the surrounding environment [25]. With the advancement of computing capabilities and access to volumes of data, machine learning models are becoming extremely popular [26]. To demonstrate the popularity and impact of this new technique in a person's life, here is a list of everyday services that uses machine learning algorithms: Email Spam Filters[27], Netflix and Amazon recommendations [28], Number plate recognition [29], Face Recognition [30], Anti-virus software [31]. Many of these models, in particular, Neural Networks are computationally intensive and are often found in large-scale data centres around the world. Data centres are viewed as particularly inhibiting towards climate goals [32]. According to the IT trade association TechUK, a large data centre can consume around 30GWh (Gigawatt hours) of power per year, enough to run 10,000 UK households [33]. Climate change targets necessitate reduction of energy consumption in all aspects, including the IT industry. It is also found that without dramatic increases in efficiency, the ICT industry could use 20% of all electricity and emit up to 5.5% of the world's carbon emissions by 2025 [34].

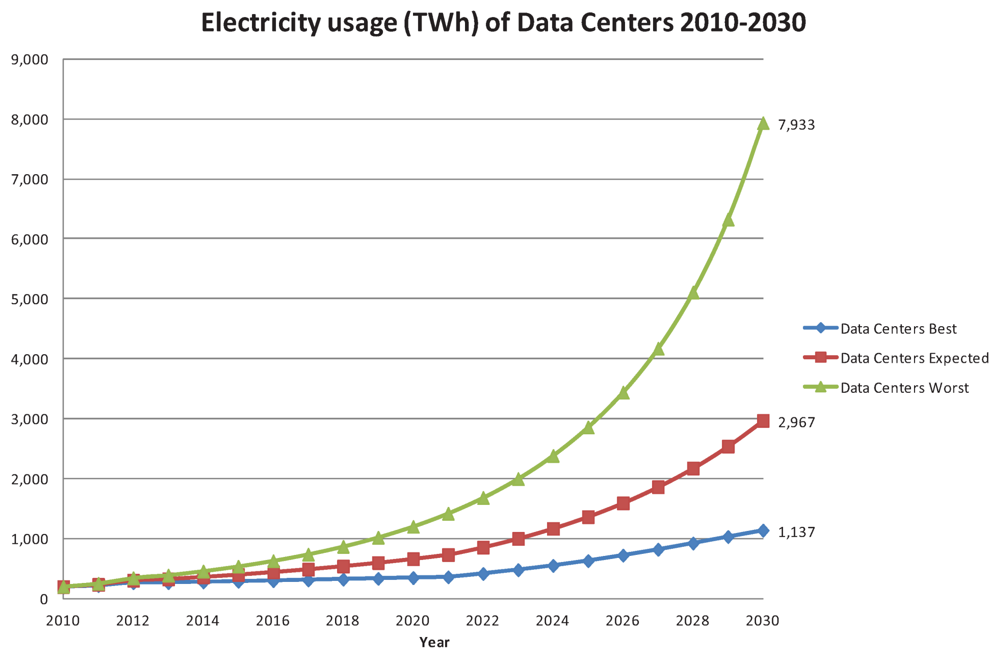


Figure 1.1: Expected energy consumption by data-centres in future [1]

Figure 1.1 shows the expected exponential growth of energy consumption in the data centres along the coming years. As we can observe, the energy usage is expected to increase multiple-folds in upcoming years. There is a clear emphasis on the need and opportunity for energy-efficiency over the coming years to make computing greener and cost-effective.

Further, these computationally intensive machine learning models have taken a toll on battery-powered devices. For example, smartphones nowadays cannot even run object classification with AlexNet (a 2012 convolutional neural network) in real-time for more than an hour [35]. Hence, energy consumption has become the primary issue of bridging machine learning models such as Neural Networks into practical applications.

Out of all machine learning algorithms, CNNs (Convolutional Neural Networks) especially have a high cost of energy consumption [36]. These Neural Networks have been successfully applied to various applications from image and video classification to speech recognition. Therefore, it is highly desirable to design frameworks and algorithms that are both accurate and energy-efficient. For the propose of this project, we would be focusing on image classification using these CNNs.

One such set of techniques reduces the energy of a given Neural Network by discriminating between given inputs. All inputs are not equal; for example, as shown in Figure 1.2, compressing a picture that has a man with a blue sky should take less effort than one that has a busy street. Ideally, to improve both speed and energy-efficiency, algorithms should spend resources (computational time and energy) that is proportional to the difficulty of the inputs. Unfortunately, for most applications, discriminating easy inputs from hard ones at run-time is challenging. Thus, hardware or software implementations tend to spend computational effort as determined by worst-case inputs [12]. To address this issue, we need to create software techniques/libraries which can dynamically discriminate on the computational needs of input to get the required output.

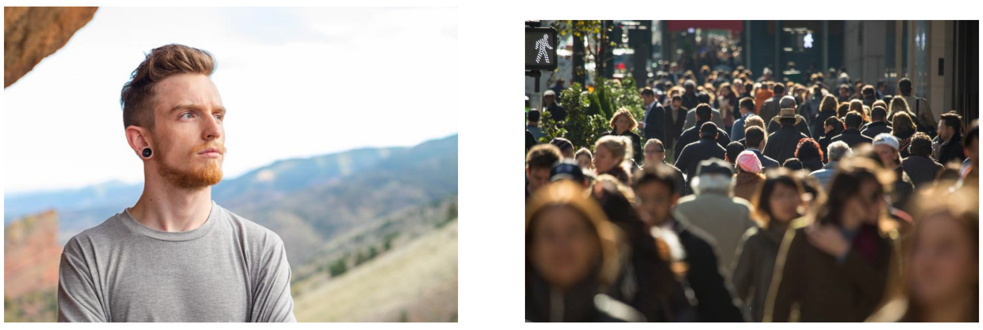

(a) Image 1 (b) Image 2

Figure 1.2: Images with different computation complexity, (a) is less computationally intensive to compress as compared to (b) [2], [3]

We look to improve the current algorithms which discriminate between given inputs by creating multiple models. These are known by different names by different popular implementations, Scalable effort Cascading Classifiers [12], conditional deep learning classifier [13], cascading neural network [37], etc. We chose this approach as it offers clear advantages over other methods. Firstly, it has very few assumptions on the underline models making it applicable to a potentially wide range of inputs types and model architectures. It doesn't require any specific hardware requirement making it easy to evaluate and use in real-world circumstances. Finally, it doesn't have a clear trade-off between accuracy and energy, making it possible to reduce energy consumption without impacting accuracy.

## 1.2 Objectives

1. **Environment to measure the energy of Neural-Networks**: The first objective of the project is to develop a reliable environment where Neural Network models can be executed and compared on different parameters such as energy consumption, accuracy, etc. We need to ensure that this environment allows a wide range of models with different data-sets to be executed. Steps also need to be taken to ensure that there is minimum interference from other applications and physical factors to provide reliable and accurate readings.

2. **Compare state of the art energy-efficient Neural Networks**: We need to examine the current software techniques to develop and design energy-efficient Neural Networks. Moreover, we hope to measure the effectiveness of some popular techniques and compare the results using the above environment.

3. **Develop and improve new techniques to design energy-efficient models**: A successful project would be able to evaluate the state of the art techniques available and extend it further to create an original method which outperforms current techniques. We should be able to then critically state both the advantages and drawbacks of this new technique along with the conditions required for it to be most effective.

4. **A Tool to generate and evaluate energy-efficient models**: Finally, we aim to develop a tool or a script for developers to apply these techniques and generate an energy-efficient Neural Network. This tool should further be able to evaluate a given model on the basis of its energy consumption.

## 1.3 Contributions

1. **Energy-efficient partial classification models with lower-dimensional input data**: We looked at different techniques to extract information from image data set with the primary goal of reducing the input complexity. We achieved this by extracting a wide range of semantic data such as colours, texture, edges, corners and using linear transformation techniques such as PCA and LDA. By reducing input complexity, we were able to reduce the model complexity making them energy-efficient. We explain this further in Section 4.1.

2. **Novel algorithm to arrange partial classification models**: We proposed an algorithm to select the appropriate partial classification models based on their results and arrange them to maximise the energy savings. We also attach a Final model that aims to accurately classify the images in case these partial classification models fail. We discuss this in Section 4.2.

3. **Energy constrained Bayesian optimisation for Final model**: To reduce and constrained energy consumption of this high-accuracy Final model, we have proposed an algorithm to perform energy-constrained Bayesian optimisation. This algorithm explores a large variety of models within given constraints to produce the model with the highest accuracy. We explain this further in Section 2.7.

4. **Script to generate and evaluate energy-efficient models**: We have exposed the code in the form of Python scripts which enable users to extract and train energy-efficient partial classification models, use the proposed algorithm to arrange these models and finally to produce Final model using energy-constrained Bayesian optimisation. We state how to use the script in Appendix A.

# Chapter 2

# Background

This chapter would cover the background research and material essential to develop this project. This part of the report will include different machine learning models in Section 2.1, the relationship between power and energy consumed in Section 2.2, tools to measure the energy consumption of models in Section 2.3. Further, we would be looking at some of the data-sets used in Section 2.4 and the state of the art research in this field in Section 2.5. Finally, we would gain some insight into the background of the techniques used in this project in Section 2.6 and Section 2.7.

## 2.1 Machine Learning Models

Machine learning is a vast field with many models and techniques. However, in this section, we will only focus on the models we have used in this project. We will be concise about the commonly known properties and would focus on the details which are lesser-known or vital to the project.

### 2.1.1 Deep Neural Networks

Deep Neural Networks (DNN) currently form the foundation of much artificial intelligence (AI) applications [38] ranging from data compression, self-driving cars, speech and image recognition [39]. DNNs can extract high-level features from raw data and have an adequate representation of input space. Then it can recognise those high-level patterns and create rules based on them. This approach is much easier than earlier strategies where experts from the field used to design these rules manually.

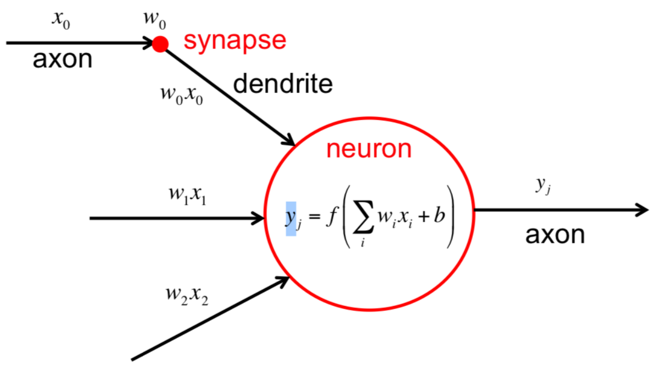
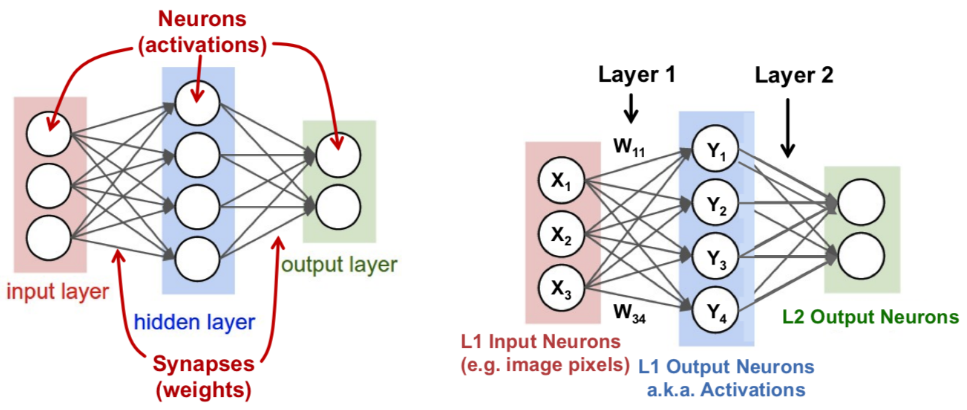


Figure 2.1: Connections to a neuron in the brain [4]
.

DNNs also referred to as deep learning, are a part of the broad field of AI, which is a science of creating the ability to achieve goals as humans do. They also fall into the category of brain-inspired computation, as the neurons present in our brain-inspired the basic structure of DNNs (Neurons). The neurons in the brain are interconnected via dendrites (entering element) and axon (leaving element). Further, each neuron performs the computation based on the input-signal (activation) and give the resulting output which is scaled based on weights (by the synapse) before passing it to the next neuron as shown in Figure 2.1. In a way, the brain learns by changing these weights between neurons and thus controlling the level of activation on a given stimulus.

Figure 2.1 also shows that the sums of the weighted activation are not added and passed as output, as this would lead to being a simple algebra operation; but instead, the result is passed to a function (called activation function) which could be linear or non-linear. Then the result of this function is passed as output.



(a) Neurons and synapses (b) Compute weighted sum for each layer

Figure 2.2: Structure of Neural Networks [4]

Neurons are often present in layers. The neurons in the input layer receive some values which are propagated further to hidden layers until finally it reaches to output layer where the result is determined. Figure 2.2 (a) shows the structure of a primary Neural Network with one hidden layer. The computation done by each neuron is shown in Figure 2.2 (b) as is given by the formulae $y_j = f(\sum_{i=1}^{n} W_{ij} * x_i + b)$, where $W_{ij}$, $x_i$ and $y_j$ are weights, input activations and output activations, f($\cdot$) is a non-linear function, respectively. Usually, Neural Networks have more than three layers, typically ranging from 5 to more than a thousand layers [5].

So to learn to perform a given task by DNNs involve learning the value of weights in the network, this task is referred to as training. The goal at this stage is to influence the weight that maximises the score (the output activation) of the correct class while minimising the score of the wrong classes. The difference is referred to as loss (L). Thus to reduce this loss, the weights are updated by a process called gradient descent given by the formulae $w_{ij}^{t+1} = w_{ij}^{t} + \alpha * \frac{\partial L}{\partial w_{ij}}$, where $\alpha$ is the learning rate. In this process, the weights are updated after every batch (a group of inputs) by this formulae until we reach a stage with a sufficiently low error rate. There are multiple ways to train weights; for this project, we would be focusing on supervised learning where the training samples are already labelled.

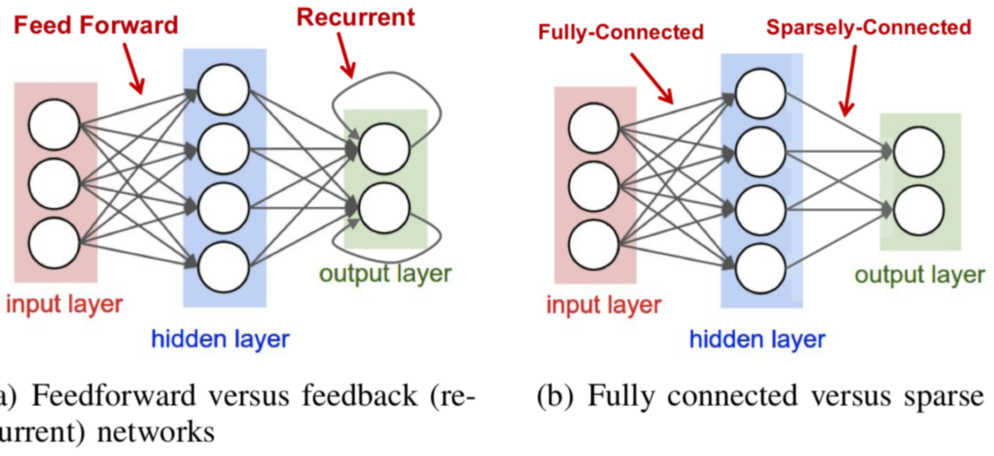


Figure 2.3: Type of Neural Networks [4]

Further, we can divide Neural Networks into two major types, feed-forward and recurrent, based on the topology of the network. In Feed-forward network, all the computation is performed as a sequence of operations of the output of the previous layer. In such a network, it has no memory of the previous input, and the output remains the same irrespective of the sequence of input. On the other hand, they have internal memory to which store data from previous input for later inputs. In this project, we would be focusing on feed-forward networks as they rely on simple weights, which also are a large part of the recurrent network.

Figure 2.3 shows the two type of networks explained above. It also illustrates of two types of layers present in DNNs, fully-connected and sparsely-connected. A Fully-connected layer has connections from all the neurons and require a significant amount of storage and computation [5]. However, some applications only require in formations from a small number of neurons; in such cases, we can use a sparely connected layer to save computation.

### 2.1.2 Convolutional Neural Networks

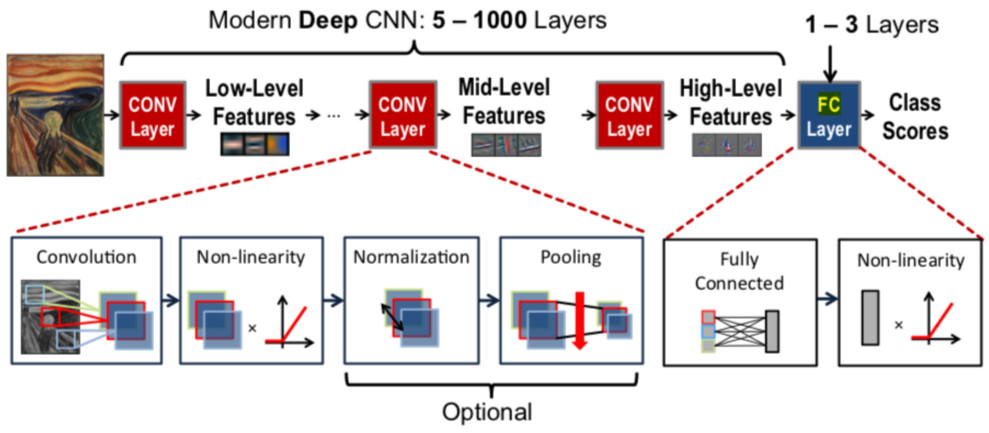


Figure 2.4: Structure of Convolutional Neural Networks [5]

Convolutional Neural Networks (CNN) are top-rated Neural Networks, especially for image recognition. CNNs comprises multiple convolutional layers, as shown in Fig. 2.4. Each of these layers contains learn-able filters which slide across the width and height of the input collecting data at every spatial position. The underlying assumption is that in a picture, the pixels nearby are correlated to each other. Intuitively, the network will learn filters that activate when they see some visual feature, such as an edge on the first layer [40]. As the layers increase, these filters will capture more and more complex features of the image, such as faces.

After these sparsely-connected connected layers, the CNNs often contain a few fully connected layers. These layers are identical to standard DNN layers; the primary purpose of these layers is to match the features extracted by the convolutional layers to the output classes present. Often, CNNs also have optional pooling layers; these are to reduce the dimensionality of a feature map to make them more efficient. Pooling combines a set of values in its receptive field into a smaller number, thus reducing the overall dimension of the layer. Finally, sometimes the architectures also use normalisation layers to improve accuracy and speed. In these, the distribution of the layer input activations is normalised such that it has a zero mean and a unit standard deviation.

We define the accuracy of a model (or a network) as the percentage of correct test samples classified when compared to the true target values.

### 2.1.3 Support Vector Machine

Support Vector Machine (SVM) is another kind of popular machine learning model used for both classification and regression. It represents the training data as data points on a plane (or space in case of more than 2 dimensions), which are divided by a hyper-plane into different categories. As Figure 2.5 explains, these hyper-planes are made as far as possible from the nearest data point to attain the best theoretical accuracy.

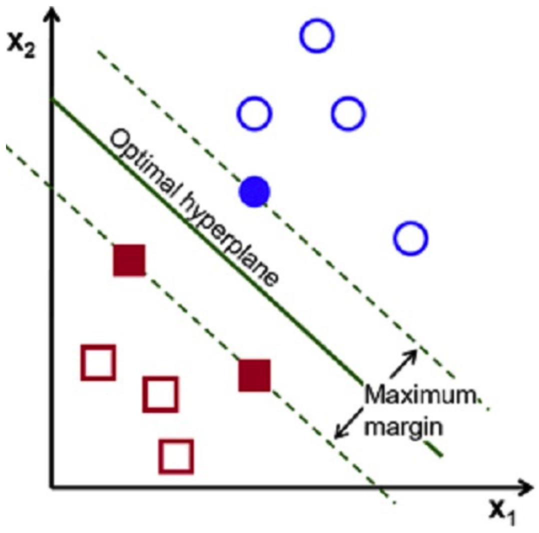


Figure 2.5: Example of Support Vector Machine Classification [6]

Often, the data is not linearly separable, and we need to use higher-dimensional kernel functions to find the more complex hyper-plane for our training data. Some examples of these higher dimensional kernels can include using polynomials and Gaussian basis functions [41]. After this training phase, the new inputs are classified by this hyper-plane to there respective classes.

The regression problem can also be solved similarly. In that case, we take a subset of training data and generate a hyper-plane with the best fit (minimum distance) from the given data. This hyper-plane is now extended to provide predictions on given test data. Figure 2.6 shows an example of a simple regression of equation of a straight line with given sample training data.

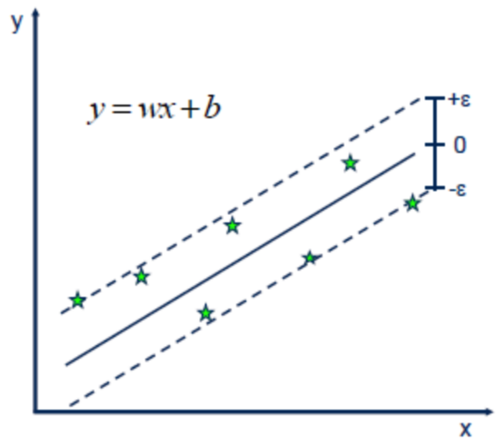


Figure 2.6: Example of Support Vector Machine Regression [7]

Advantages of using SVM include kernel trick: with an appropriate kernel function, we can solve any complex problem [42]. It scales relatively well to high dimensional data. Finally, memory efficient as it may use only a subset of training points to compute decision function. On the other hand, the main disadvantage is that it is often hard to choose a kernel model suitable for a given problem, they do not provide probabilistic estimates for classification problem, and difficulty in optimising regularisation and kernel function to avoid over-fitting.

## 2.2 Relationship between power and energy

Often power and energy are used interchangeably; however, power is defined as the rate of doing work, measured in Watts [43]. This can be precisely be given with the following formulae below.

$$energy = power * time$$

Energy (joules) is the product of power (watts) and time (seconds).

For a given program, we can also calculate the average total energy consumed by the program by the given formulae.

$$E = IC * CPI * EPC$$

The total energy of a program (multicycle computer model) is the multiplication of the number of instructions (IC), the average number of clock cycles per instruction (CPI), and Energy per cycle (EPC). The key is that CPI is different for the types of instructions; thus, certain instructions need more CPI than others. For example, ALU instruction has a lesser CPI than load instruction. A way to optimise a program to consume less energy and to take less time is to reduce the average CPI of the program instructions [26].

Within this report, we have used energy, power and average power interchangeably. The goal of the project is to reduce the overall energy consumed by the algorithms.

## 2.3 Tools for energy Measurement

The above approach can be a good starting point to measure energy consumption; however, for convenience and better accuracy using energy measurement tools would be ideal for our purpose.

### 2.3.1 PowerKap

A previous Imperial student developed this tool as his final year project [8]. The main Idea of PowerKap is to build a profiling mechanism external to the application/program running. This approach has numerous advantages being that PowerKap does not interfere with the security features of the application. Moreover, this makes it compatible with a large number of applications without any modification required in the application code itself.

PowerKap works by sampling counters during the execution of a program at regular intervals. The user program annotates the points of interest and marks them by printing timestamps during execution. This process is repeated multiple times to solidify the gathered measurement further. PowerKap also explicitly controls the execution of the child program, giving it the added benefit of greater time-control. This control allows PowerKap to be aware of the times the child program is active, leading to more effective and accurate results [8]. The structure of PowerKap is given in Figure 2.7.

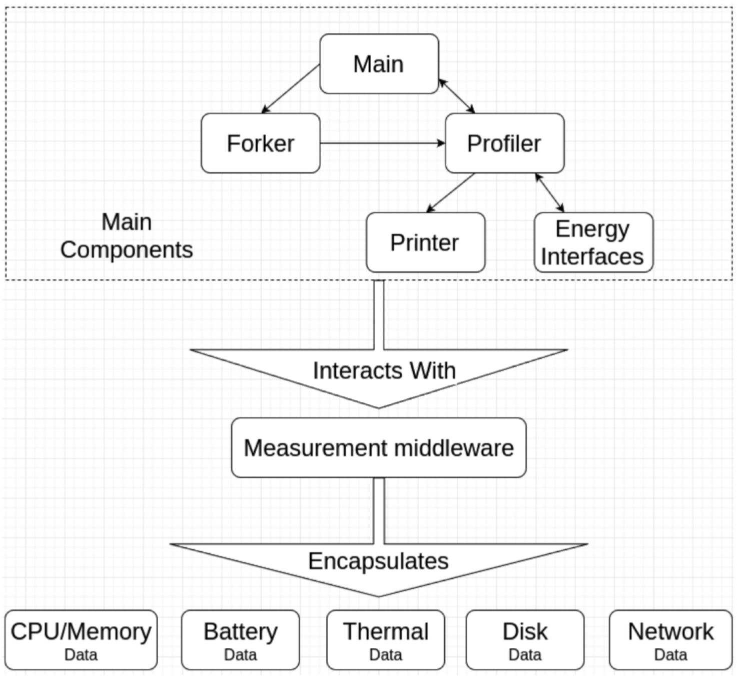


Figure 2.7: A simple overview of PowerKap [8]

Another main advantage of using PowerKap could be its ability to gather data from various sources and interfaces available in the user-space. This includes energy consumption from CPU and RAM in the form of RAPL interface, DiskIO and Networking. Although for our project, CPU and Ram energy consumption would be likely most relevant, factoring in other information could be useful for further extensions.

**RAPL (Running Average Power Limits) Interface**

On Intel platforms, energy counters are accessible whiten the CPU directly. This is provided in the RAPL interface which is specific to certain areas (refereed as domain) of the computer.

1. **Package** The first domain available is package and is generally for supporting multiple CPUs. The statistics generated in this layer are specific to the CPU package as a whole. The measurements are taken place every millisecond; however, this can vary depending on the load on the CPU loads [8].

2. **Core (PP0)** This refer to the CPU core with L1 and L2 cache within a package. The stats are generated based on the energy counter, which is updated every millisecond [8].

3. **Uncore (PP1)** This interface is identical to the Core interface above and is generally present in the consumer devices. The Uncore is generally referred to a portion of the chip outside Core such as L3 cache rings, Integrated GPU and memory controller.

4. **DRAM** Intel offer additional capability (Generally on Server and Certain architecture) to capture energy consumption of the memory controller.

The PowerKap tool outputs the CPU and RAM energy consumption in the form of RAPL interface. It's quite helpful as this is an open-source and popular standard.

### 2.3.2 Intel Power Gadget

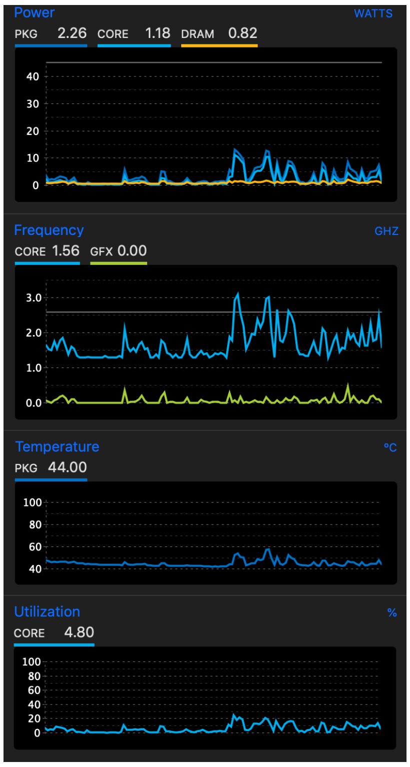


Figure 2.8: A sample GUI result of PowerGadget tool

Intel Power Gadget is a software-based energy usage monitoring tool for Intel processors (from 2nd Generation up to 7th Generation). This includes an application driver, and libraries to monitor and estimate real-time processor power information in watts. This information is calculated using the energy-counters in the processor. The motivation for the tool was to assist end-users, Independent Software Vendor (ISV)s, developers, and others interested in a more precise estimation of power from a software level without any H/W instrumentation [44]. The following Figure 2.8 shows the GUI of the tool. The results can also be logged in a file for in-depth analysis.

### 2.3.3 s-tui and psutil

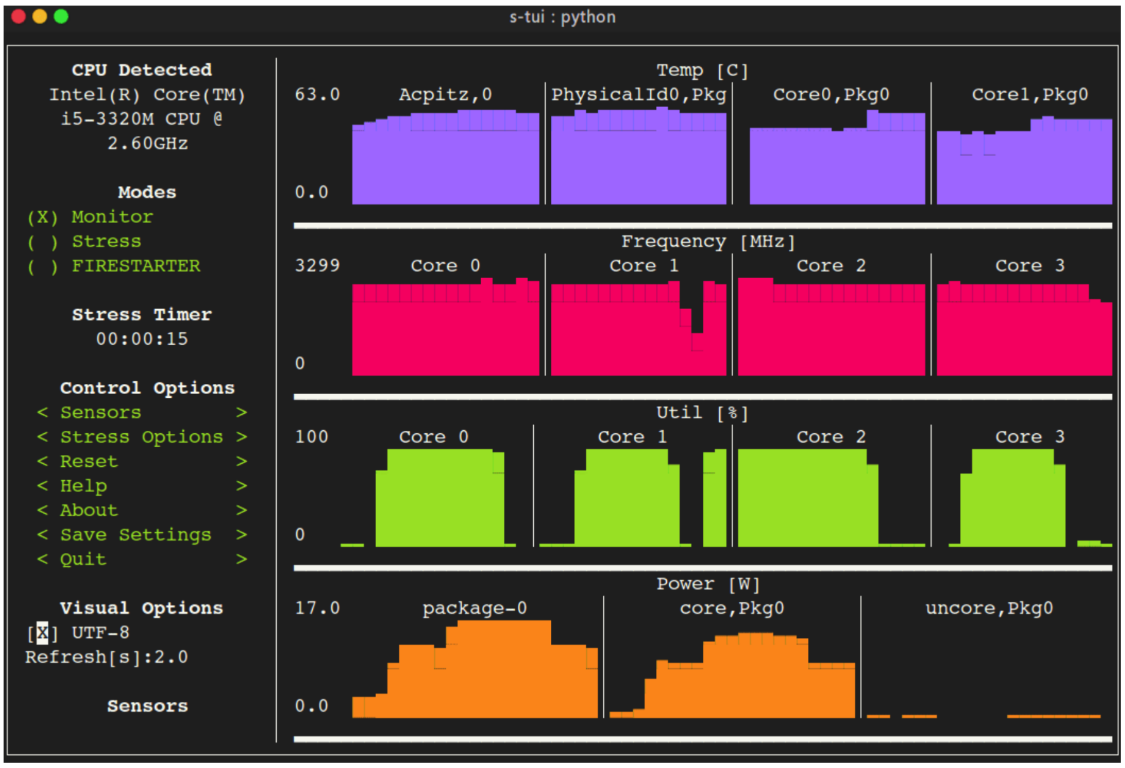
#### psutil

psutil (python system and process utilities) is a cross-platform library for retrieving information on running processes and system utilisation (CPU, memory, disks, network, sensors) in Python [45]. It is useful mainly for system monitoring, profiling, limiting process resources and the management of running processes [45].

It provides the functionality to get the current status of CPU, Memory, Disk, Network and external sensors along with platform-specific information about each. This helps track the user important information such as in the case of CPU, it helps track the physical cores usage, cpu_frequency, etc. information which could be useful to understand the underline energy usage.

#### s-til

s-tui [46] or Stress Terminal UI provides a simple interface to monitor the CPU and RAM information. It uses the psutil library and the RAPL interface provided by Intel as discussed in Section 2.3.1. This RAPL interface provides detailed information about the energy usage by each package at every second.


```
CPU Detected
Intel(R) Core(TM)
i5-3320M CPU @
2.60GHz

Modes
(X) Monitor
( ) Stress
( ) FIRESTARTER

Stress Timer
00:00:15

Control Options
< Sensors         >
< Stress Options  >
< Reset           >
< Help            >
< About           >
< Save Settings   >
< Quit            >

Visual Options
[X] UTF-8
Refresh[s]:2.0

Sensors
```

Figure 2.9: A sample GUI result of S-tui tool

The power stats is provided by Intel in the file ('/sys/class/powercap/intel-rapl:*/') in RAPL format. This information is parsed, appended and provided in both the UI and logs by this library. Further, it uses information from the psutil to mark active cores, temperature and frequency to give a piece of comprehensive information about the net energy utilisation.

Figure 2.9, shows the GUI result of the s-tui library. For our implementation, we would ignore the GUI and run the library in the background. The energy information would be stored in logs which would be parsed to determine the net energy consumption.

## 2.4 Data-Sets

Since one of the goals of the project is to compare the energy-efficiency of multiple CNN models; we chose one of the most popular and open-source data-sets available. We have also chosen databases with relatively smaller in size with fewer classes to classify for convenience as the goal of the project is to illustrate concepts rather than build a sophisticated Neural Network.

### 2.4.1 MNIST

MNIST is one of the most popular deep learning datasets which is a subset of a more extensive set available from NIST. It's a dataset of handwritten digits that contains a training set of 60,000 examples and a test set of 10,000 examples. The digits have been centred and size- normalised in a fixed-size image of 28*28 pixels. This pre-processed dataset allows developers to quickly test learning techniques and pattern recognition methods with minimum effort [47].

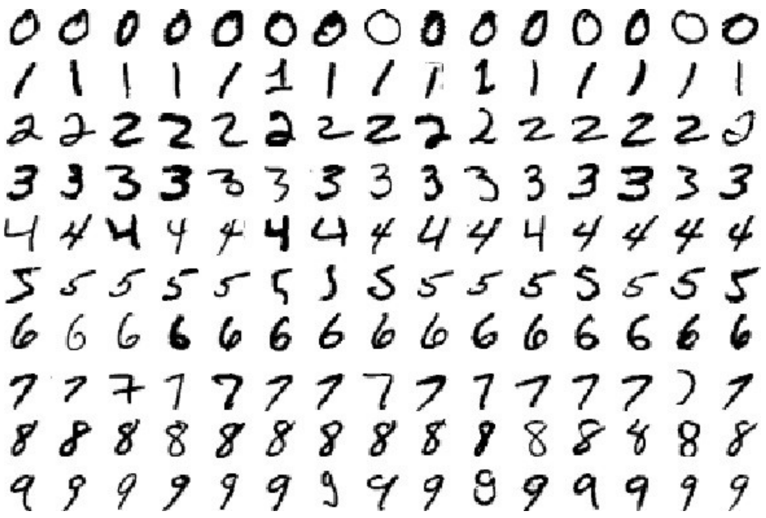


Figure 2.10: MNIST dataset [9]

There have been several scientific papers which use this data-set for there research, in-fact even Tensorflow and Keras allow us to import and download the MNIST data-set directly from their API.

### 2.4.2 CIFAR-10

CIFAR-10 is also one of the popular datasets for deep learning algorithms. CIFAR-10 dataset consists of 60,000 32x32 colour images in 10 classes, with 6000 images per category.

These 60,000 images are again split as 10,000 test images and 50,000 training images. The 10 different classes represent aeroplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. All of these images exclusively belong to one of the classes, meaning there is no overlap between the classes for a given input [48].

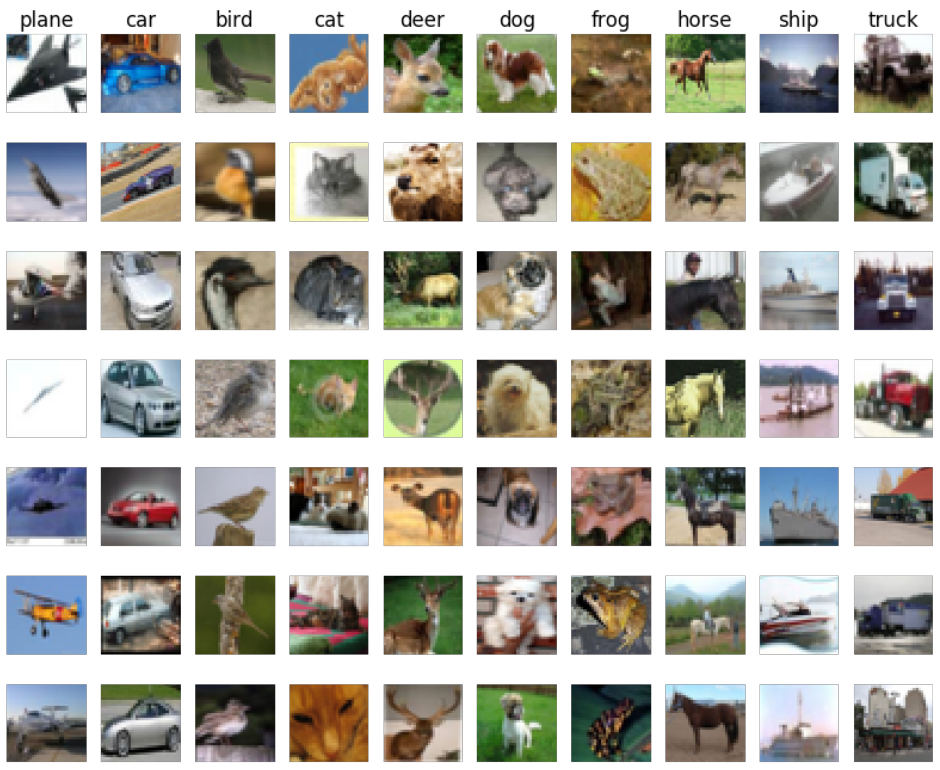


Figure 2.11: CIFAR-10 dataset [10]

### 2.4.3 Caltech 101

Caltech 101 is a data set of digital images compiled by Fei-Fei Li, Marco Andreetto, Marc 'Aurelio Ranzato and Pietro Perona at the California Institute of Technology. It is intended to facilitate Computer Vision research and techniques and is most applicable to techniques involving image recognition classification. It has a total of 9,146 images, split between 101 distinct object categories with about 300x200 pixels - 40 to 800 images per category with most categories having about 50 images. Common and popular categories such as faces tend to have a larger number of images than others. The main advantage of this data-set is that almost all the images are uniform in image size and the relative position of interest objects. Moreover, users do not need to crop or scale images before using them [49].

## 2.5 State-of-the-art energy optimisations for Neural Networks

In this section, we would discuss some of the recent papers related to this subject. The basic idea or each approach is explained briefly skipping over the minor implementation details. We have divided them into four main sections; Firstly, we would discuss the hardware approach where researchers are trying to optimise the hardware to run a certain kind of Neural Network. Secondly, we detail approaches to make low-level approximations to get better results, followed by a few established techniques to create multiple scalable models for a given model. Finally, we would be discussing pruning technique to improve the energy-efficiency of the given model.

### 2.5.1 Hardware Accelerators

Much research in this field is done in hardware-accelerators. These accelerators are generally designed to improve the speed and efficiency of specific instructions which are often used while training and testing Neural Networks.

One such accelerator is Low-Rank Approximation for efficient Deep Neural Network (LRADNN). This accelerator bypasses the inactive neuron calculations in a DNN to only account for weights of neurons which are predicted to be active, which allowed a lesser number of arithmetic operations with only a small performance degradation. They also developed a Low-rank approximation (LRA) predictor, which calculated these synaptic weights based on accuracy and computation complexity. Finally, they implemented this proposed accelerator with a 5-stage pipeline architecture using TSMC 65nm technology. They had impressive results with up to 53% energy saving and 43% throughput increase for specific architectures [50].

Another interesting approach could be taken by optimising the data-flow to improve the Neural Network Accelerators. Memory access usage and data movement often consume most of the energy in a DNN. If we can maximise this by re-using the data and storing it in low-energy-cost storage levels, then we can dramatically improve the energy-efficiency of the Neural Network. The paper argues that there are many opportunities for data re-use, which include use filter re-use, wherein each filter weight is re-used across multiple input feature map. Moreover, we could re-use the partial sums calculated for each output feature map. Finally, the paper proposed a new technique called Row-Stationary (RS), which optimise the data movement for superior system energy-efficiency. This evaluated to be up to 2.5 times more energy-efficient than existing data flows in processing a state-of-the-art DNN. There are also guidelines provided for future work in this field in the paper [51].

RAPIDNN is another proposed framework to solve the data movement energy consumption problem described above. The proposed memory design is capable of supporting all the DNN functions in memory itself. This is extending the idea of processing in memory (PIM), where the data is not sent to the cores of the computer to be processed but is partially processed in the memory itself. RAPIDNN first interprets the DNN model and maps all the operation in the specialised accelerators (which support PIN operations with non-volatile memory). Then at run-time, the accelerators compute the result using these mapped operations in-memory to provide the required solution. This approach achieved impressive results as well with 68.4 times energy-efficiency improvement and 48.1 times speed improvement, with almost no loss in quality [52].

Although these hardware approaches are providing impressive results, we have decided to lean towards software approaches for our project. The reason for this is many-fold, firstly, since a lot of research is already taking place in this field, it might be harder to find a novel solution. Secondly, it would be hard to design and test any proposed solutions due to lack of resources and knowledge in this particular field. Finally, it is often hard to replicate any established approach without having specific hardware support.

### 2.5.2 Approximate Computing

Approximate computing or Stochastic Computing is another effective method to reduce the computation complexity and energy consumption, especially in the 'light-weight' hardware. The basic idea is only to compute numbers to a given significant precision. This leads to saving on computation but include random error fluctuation in the results.

Typically existing approaches commonly fall into three categories: the first being application-specific hardware or software, which take advantage of low-complexity algorithms coupled with hardware tailored to the application-specific needs. One such approach is dynamically removing near-zero weights, applying weight-scaling, and integrating the activation function with the accumulator until the required energy savings have been achieved. This is done on fixed hardware that supports the needed characteristics of stochastic computing [53].

The second being the design of application-independent hardware systems, one such example would be to approximate Multiplier Design using bit significance-driven logic compression [54]. They designed a new multiplier design which systematically compresses lower-significance bits of the highest order multiplier to achieve substantial energy savings at a low loss of accuracy. This also reduces the resulting number of product terms, which result in lower compute cycles for the same multiplications.

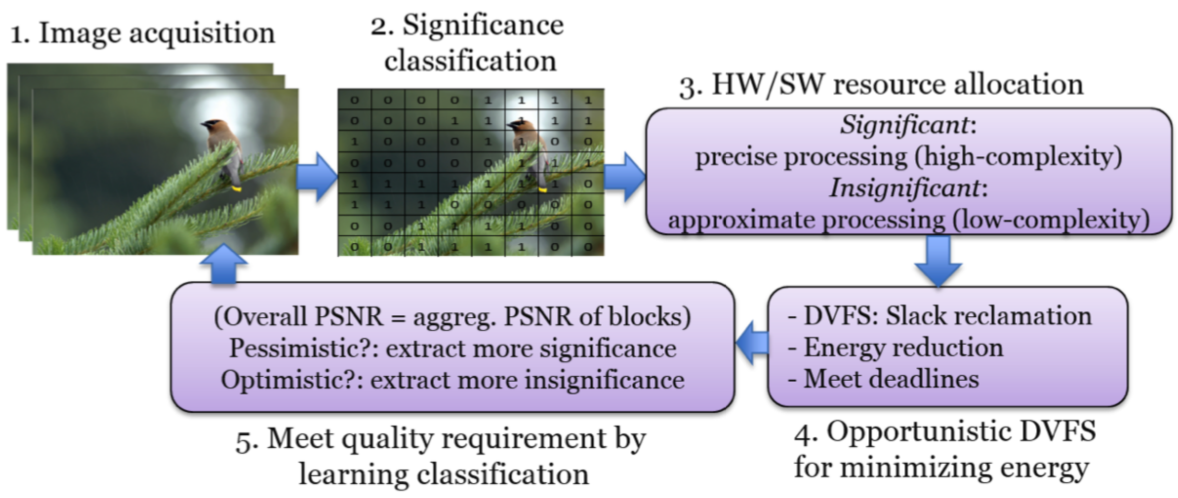

1. Image acquisition
2. Significance classification
3. HW/SW resource allocation
Significant: precise processing (high-complexity)
Insignificant: approximate processing (low-complexity)
- DVFS: Slack reclamation
- Energy reduction
- Meet deadlines
4. Opportunistic DVFS for minimizing energy
(Overall PSNR = aggreg. PSNR of blocks)
Pessimistic?: extract more significance
Optimistic?: extract more insignificance
5. Meet quality requirement by learning classification

Figure 2.12: Image approximation technique [11]

Third and the last type would be to approximate the input values itself and to use hardware and software solution to adapt depending on the significance of the input. One such approach defines the relevance of the image through parallel inference of mean and standard deviation per image block (a small section of the image) [11]. This is done in a loop until we meet the required energy standards by accurate processing of high-complexity areas and approximate processing of lower complexity regions, as shown in Figure 2.12. The underpinning evaluation based on the significance is performed on the given hardware and software implementations.

The main advantage of this approach is that it can often result in significant improvements; especially in the specific cases where a computation method is used multiple times during processing. On the other hand, by design, we make clear trade-offs between accuracy and energy-savings of the model and therefore, the only way to make significant energy savings is to accept a significantly lower precision. Moreover, most approaches in this domain require implementing/manipulating low-level hardware implementations, which would be hard to test and verify for this project, given the restrictions in knowledge and resources.

### 2.5.3 Scalable effort Cascading Classifiers

The classification problem is one of the significant and widespread applications for machine learning algorithms. Scalable effort classifiers provide a new approach for optimised, more accurate and energy-efficient supervised machine-learning. The main idea of this approach is to generate multiple models with increasing complexity instead of one complex model for classification. This allows us to discriminate on which model to use based on the complexity of the input. We can, therefore, use an easy model to classify test inputs with low complexity. On the other hand, we would need to use the complex model for the test inputs with higher complexity.

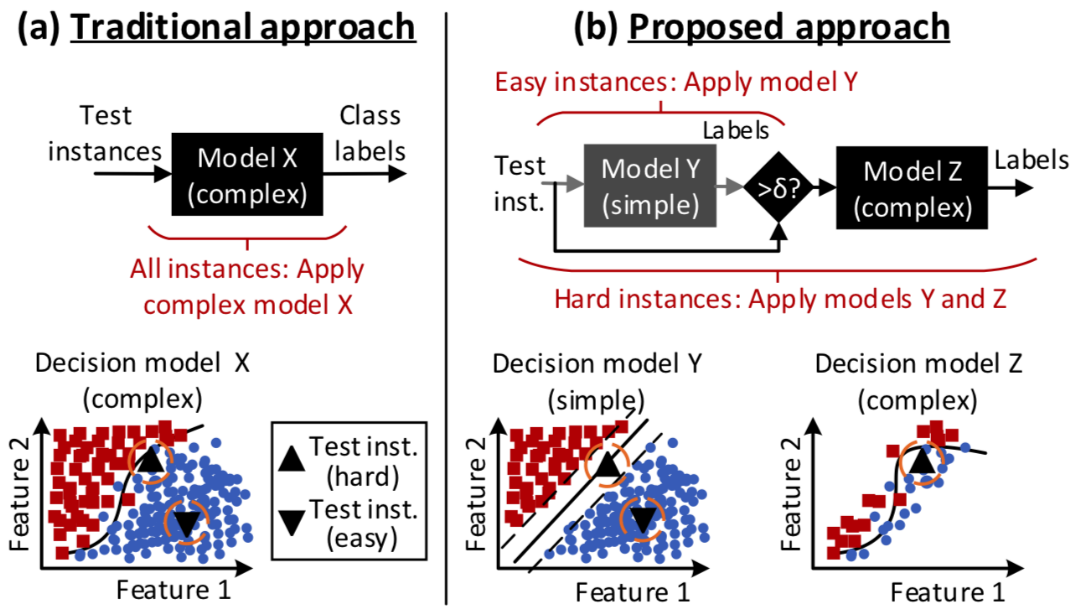
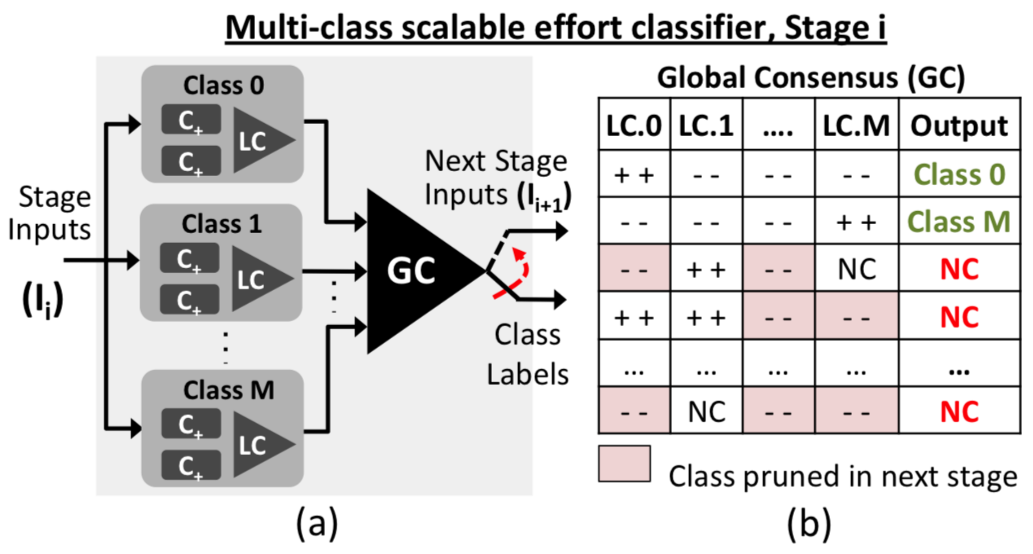


Figure 2.13: Basic approach by Scalable effort Classifiers [12]

Figure 2.13 explains the approach using an SVM (Support Vector Machine) example. In the traditional model, we would classify both easy and hard test input with the same complex SVM model. This resulted in energy-intensive and expensive computation even in the cases when not required for many inputs. In the proposed, approach we train two models, we first pass all the inputs on the simpler model and check the confidence interval. Then only the hard test input (ones closer to classification hyper-plane in this case) are applied to the complex model. This technique has provided up to 2.3× and 1.5× improvement in energy and run-time [12].

Another significant contribution of the paper is the method to determine the complexity of the inputs at run-time. This was a substantial bottleneck for scalable classifiers before this approach was introduced. It includes passing the given test inputs on the simpler model and checks the confidence level of the output. To measure the confidence level, we train biased classifiers for each class of the output. These biased classifiers are specialised in predicting the presence of one class of the output. After this, we apply a consensus algorithm to determine the confidence level of the model. If the global consensus, as in the case of Figure 2.14, is formed when the model is classified otherwise, it is sent to the next more complex model to repeat the process.



| LC.0 | LC.1 | .... | LC.M | Output |
|---|---|---|---|---|
| + + | - - | - - | - - | Class 0 |
| - - | - - | - - | + + | Class M |
| ▨ - - | + + | ▨ - - | NC | NC |
| + + | + + | ▨ - - | ▨ - - | NC |
| ... | ... | ... | ... | ... |
| ▨ - - | NC | ▨ - - | ▨ - - | NC |

▨ Class pruned in next stage

Figure 2.14: Determine the complexity of the model at run-time using biased classifiers and Global Consensus [12]

Further, there have been advancements in the conditional Deep Learning Classifier [13], Cascading Neural Network [37], the Distributed Deep Neural Network [55]. These methods primarily try to converge all these above models into one model with a different termination clause after each layer. Evidently, in such cases, we would have to test the confidence interval after each layer to check if it meets the threshold. If the threshold is met, we exit the model with a unique output layer at each stage, as shown in Figure 2.15.

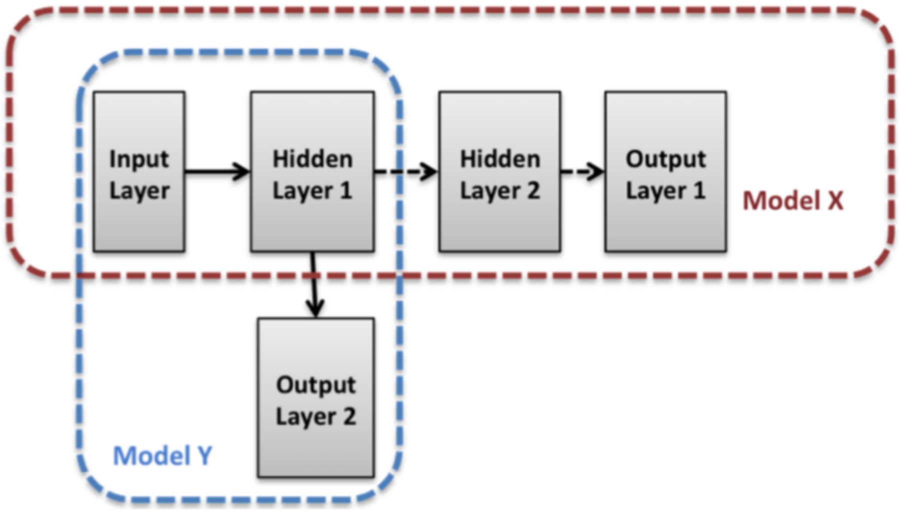


Figure 2.15: Illustrating conditional deep learning classifier [13]

However, it is shown in a recent study [56], the layer level manipulation and execution like shown above are far out-performed by the network layer adaption. Therefore, we would be focusing on network-level designs. Big/Little Deep Neural Network [57], focuses on the similar network-level approach where it creates the concept of 'confidence level' of the network, a score given by initial network to decide if we would use further networks is theorised and further solidified.

Finally, there are tree-structured approaches [58] and [59], where a hierarchical Deep

Neural Network is used in a tree structure with CNNs at multiple levels. These approaches are promising as they allow to dynamically select the correct Neural Network at each stage based on its complexity and domain of possible output classes. However, we could argue that the algorithms and models used to in this type of approach are trivial and we would examine them in further to improve their effectiveness.

The main advantages of this approach that it is general, i.e. it can be applied to a wide variety of machine learning models. The above papers also describe a basic algorithm to train these scalable models and process the test inputs for the same, which gives us a good start point. A clear trade-off between accuracy and energy conservation is also not evident, in-fact in some instances like [12] we can make the system both more accurate and energy-efficient at the same time. Further, unlike other approaches, there is no requirement for specialised hardware to perform these approaches. On the other hand, we make certain assumptions about the given model and training data; firstly, we assume a wide range of complexity of inputs in the test and training datasets. Moreover, we also assume each output should belong to one output class.

### 2.5.4 Energy-Aware Pruning

The concept of pruning is to remove lesser-used parts of Neural Network in order to improve performance and energy-efficiency with some loss in the accuracy of the network. Multiply-and-accumulate (MAC) operations in convolutional layers and fully connected layers dominate the total operations performed in the state-of-the-art CNNs and therefore form a large part of the processing run time and energy consumption. The energy consumption of these operations comes from computation and memory accesses for the required data both in feature map and weights. Simply reducing the number of weights or operations in CNN might not have a major impact on energy consumption because fetching data from memory consumes orders of magnitude higher energy than the computation itself [14]. The energy consumption by the memory for given weights is quite unpredictable as weights, and input activations are often reused and therefore can be stored in the cache, which is much more faster and energy-efficient. As a result of this uncertainty of energy consumption based on hardware and CNN design, the proposed technique uses the energy measurement directly to make pruning decisions rather than a traditional weight-based approach.

The first step of the process is to choose which layer to prune. As more and more layers are pruned, it becomes increasingly challenging to prune weights because the accuracy approaches the given threshold. Thus, we should start with the layer, which takes the most energy to maximise our savings. The next step involves removing weights less than a given threshold. This step might result in some loss in the accuracy and some imbalance of weights between layers. The third step includes restoring the weights to gain back some accuracy; this is done by solving an L0 optimisation problem. We make sure that only a certain number of weights are set back to prevent the algorithm from restoring the weights in small residues [14].

Finally, in the fourth step, we perform a local optimisation to the entire layer. This can involve changing all the weights in the layer to maximise the accuracy of the layer. These four steps are repeated until all the layers are pruned. Finally, a global fine-tuning is performed across the entire network. In this step, all the residue weights are changed to maximise the network accuracy (similar to the last step) across the entire network. Figure 2.16 describes the steps in a flowchart format.

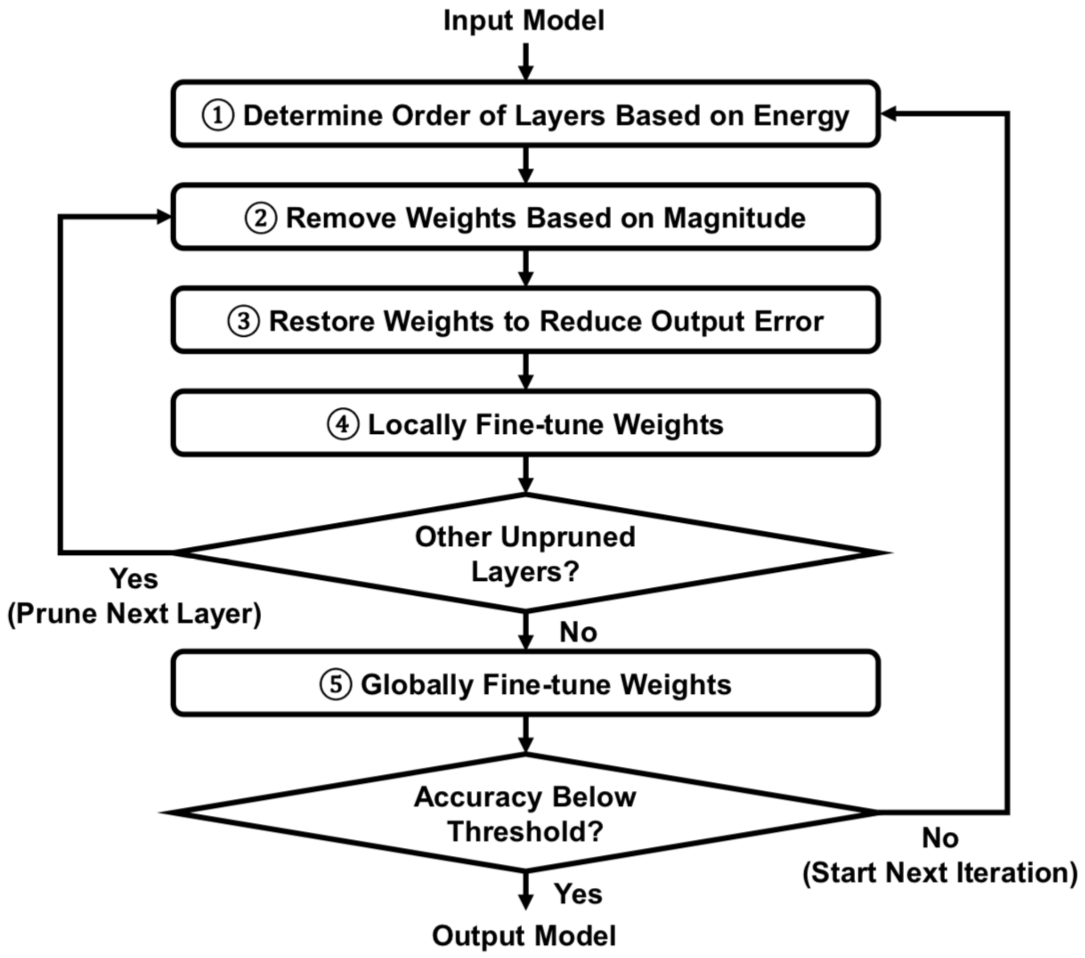

Input Model
① Determine Order of Layers Based on Energy
② Remove Weights Based on Magnitude
③ Restore Weights to Reduce Output Error
④ Locally Fine-tune Weights
Other Unpruned Layers?
Yes (Prune Next Layer)
No
⑤ Globally Fine-tune Weights
Accuracy Below Threshold?
No (Start Next Iteration)
Yes
Output Model

Figure 2.16: Steps of the proposed Energy-Aware Pruning technique [14]

An advantage of this given technique is that it could be applied to any CNN or potentially any Neural Network. Moreover, it can improve the energy-efficiency of a given trained state-of-the-art network aggressively (AlexNet and GoogLeNet are reduced by 3.7× and 1.6×). The main disadvantages include the clear trade-off between accuracy and efficiency, as effectively we are moving towards a simpler model which provides energy savings at the cost of reducing complexity and accuracy in potential edge cases. Moreover, it is likely to be only applicable to large Neural Networks with many layers and neurons, and we observe it is not useful in relatively more straightforward models.

## 2.6 Dimensionality Reduction

It is to be noted that when we use the term 'dimensionality', we do not only refer to the number of parameters or properties but can often also refer to the number of density of these inputs parameters well. For example, setting many non-relevant (less relevant) parameter values as 0.

### 2.6.1 Statistical Techniques

There are two popular linear transformation techniques to reduce the dimensionality of large data-sets.

## PCA

Principal Component Analysis, or PCA, is dimensionality reduction technique that transforms a large set of variables (dimensions) into a smaller one without losing most of the information (variance) provided by a large set of dimensions. It is to be noted that PCA does not select a set of features and discard other features, but it infers new features, which best describe the data-set.

The first step of the process is the standardisation of that data to bring all the data on the same scale. This is similar to pre-processing or normalisation of data and is primarily done to prevent a small range of values from dominating the calculations. After this step, we construct a covariance matrix of p $\times$ p (where p is the number of dimensions). It helps define the relationship between the dimensions. An example of 3 dimension covariance matrix is given below.

$$M = \begin{bmatrix} Cov(X,X) & Cov(X,Y) & Cov(X,Z) \\ Cov(Y,X) & Cov(Y,Y) & Cov(Y,Z) \\ Cov(Z,X) & Cov(Z,Y) & Cov(Z,Z) \end{bmatrix}$$

where

$$Cov(X,Y) = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{n-1}$$

Next, we calculate the eigenvectors and eigenvalues of this covariance matrix. Eigenvectors resemble the directions of the axes where there is the most variance (most information), and eigenvalues resemble the scalar with which it will be stretched after the transformation. We, therefore, extract the k eigenvectors corresponding to k-largest eigenvalues. These are the new dimensions of the data.

In the final step, we cast the original data to use the feature vector formed using the eigenvectors of the covariance matrix. To achieve this, we reorient the data from the original axes to the ones represented by the principal components (given by the most significant eigenvectors).

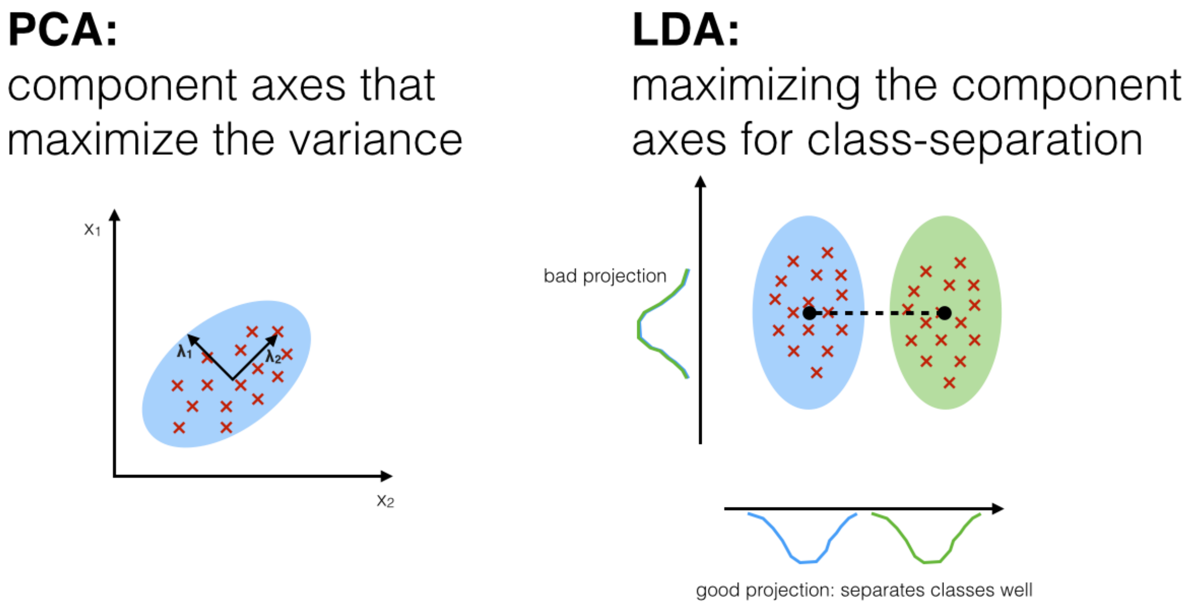

PCA:
component axes that maximize the variance
x1
x2
λ1
λ2
LDA:
maximizing the component axes for class-separation
bad projection
good projection: separates classes well

Figure 2.17: Illustrates the basic difference between PCA and LDA [15]

**LDA**

Similar to PCA, LDA or Linear Discriminant Analysis is a dimensionality reduction technique aimed to preserve most of the information (variance). However, unlike PCA, LDA is a supervised data compression technique and is also aimed at increasing class distinction (distance between the classes). Therefore a good LDA seeks to find the feature subspace that optimises class separability and maximum variance in a data-set.

The first step for calculating LDA is the standardisation of data similar to PCA. The next step is finding the mean for each class of the output. We use this to calculate class scatter-matrix, and the between-class scatter matrix. This can be calculated by the following formulae.

$$M = S_w^{-1} S_b$$

$$S_w = \sum_{i=1}^{c} S_i$$

$$S_b = \sum_{i=1}^{c} N_i (m_i - m)(m_i - m)^T$$

where

$$S_i = \sum_{x}^{n} (x - m_i)(x - m_i)^T$$

In the formulae above $S_i$ denotes the variance of the data for any particular class. $S_w$ denotes the within-class scatter matrix and is the sum of the variances over all the classes. We seek to minimise the overall sum of all the variances together. $s_b$ denotes the between class scatter-matrix with as $m$ as the mean of the classes. We also aim to maximise this to increase the distance between the output classes. We use these to calculate the net covariance matrix ($M$).

The final step is similar to PCA where we cast the original data to use the feature vector formed using the eigenvectors of the largest eigenvalues of the covariance matrix. Figure 2.17 illustrates the fundamental difference between PCA and LDA.

### 2.6.2 Semantic Techniques

Often overlooked is the amount of semantic information present in complex inputs such as images, colour, texture, edges and corners. This information can be extracted and often used in place of actual images to perform classification on a higher level. One of the popular approaches used is to extract the HSV Colour space of the images and then using popular clustering techniques such as K-mean to cluster the images into groups [16], [60].

In the traditional K-mean clustering algorithm can be computationally intensive but extracting the colour spaces and executing the algorithm, as shown in Figure 2.18, saves a lot of computation complexity which in-turn can save a large amount of energy. However, it is to be noted that choices of colour space may have significant influences on the result of image segmentation– Therese many kinds of colour space, including RGB, YCbCr, HSV. Although RGB and YCbCr colour spaces are commonly used in raw data and coding standards, they are not close to human perceptions and often do not yield correct results in the classification of objects. Therefore the choice of semantic play a significant part inefficiency of the algorithm.

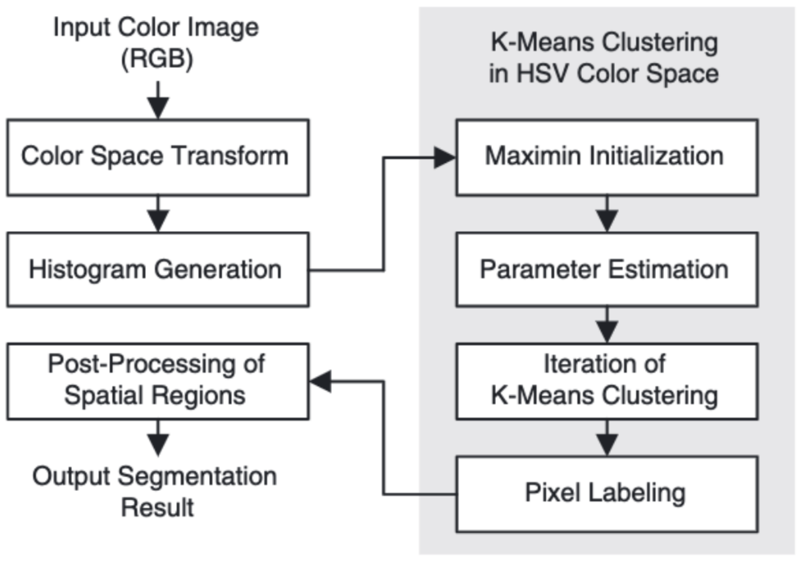

Input Color Image (RGB)
Color Space Transform
Histogram Generation
Post-Processing of Spatial Regions
Output Segmentation Result
K-Means Clustering in HSV Color Space
Maximin Initialization
Parameter Estimation
Iteration of K-Means Clustering
Pixel Labeling

Figure 2.18: Steps of the proposed Semantic K-mean clustering algorithm [16]

Furthermore, semantic information also used widely used in use cases such as objection detection. In such cases, this information is useful to train models to classify out the negative examples without spending a large amount of computation and energy. One such example is Object Detection using Semantic Decomposition [61], uses colour/texture information from images to classify out the negative objects images using a simple Neural Network.

## 2.7 Bayesian Optimisation

Bayesian optimisation is a class of machine-learning-based optimisation methods that aims to solve a maximisation or a minimisation problem of the following form.

$$max_{x \in A} f(x),$$

In the above equation, input x is in $R^d$ for a value of d (dimensions) that is not too large. The set A is bounded set where x can take any values with dimension d from that set. Also generally, we assume that the $f$ function is continuous, np-hard and “expensive to evaluate”. This also implies that $f$ is not of the form of particular form such as of from polynomial, trigonometric, etc. as there are specific techniques that leverage such structure to improve efficiency.

Further, Bayesian optimisation is only useful in cases where f performs as a “black-box”, and we observe only $f(x)$, with information such as first-order and second-order derivatives are not known. Also, for the scope of this project, we would also assume that the $f(x)$ produce no noise or noise of negligible value. Finally, we are aiming to find the global optimum rather than a local solution.

The ability to optimise expensive derivative-free black-box functions makes this technique extremely useful. One of the recent widespread use of this technique is tuning of hyper-parameters in machine learning algorithms, especially Deep Artificial Neural Networks [62]. Other uses include calibration of environmental models [63], and in the optimisation of reinforcement learning [64].

### 2.7.1 Basic Algorithm

There are two main parts for the Bayesian optimisation algorithm; a Bayesian statistical model to model the given objective function and an acquisition function to decide which point to sample next in the model. Initially, the algorithm starts with by choosing and measuring the value of a few samples randomly. Then further new points are determined based on the acquisition function until we reach the budget N as shown in Figure 2.19.

**Algorithm 1** Basic pseudo-code for Bayesian optimization

```
Place a Gaussian process prior on $f$
Observe $f$ at $n_0$ points according to an initial space-filling experimental design. Set $n = n_0$.
while $n \leq N$ do
    Update the posterior probability distribution on $f$ using all available data
    Let $x_n$ be a maximizer of the acquisition function over $x$, where the acquisition function is computed using
    the current posterior distribution.
    Observe $y_n = f(x_n)$.
    Increment $n$
end while
Return a solution: either the point evaluated with the largest $f(x)$, or the point with the largest posterior
mean.
```

Figure 2.19: Steps for Bayesian Optimisation algorithm [17]

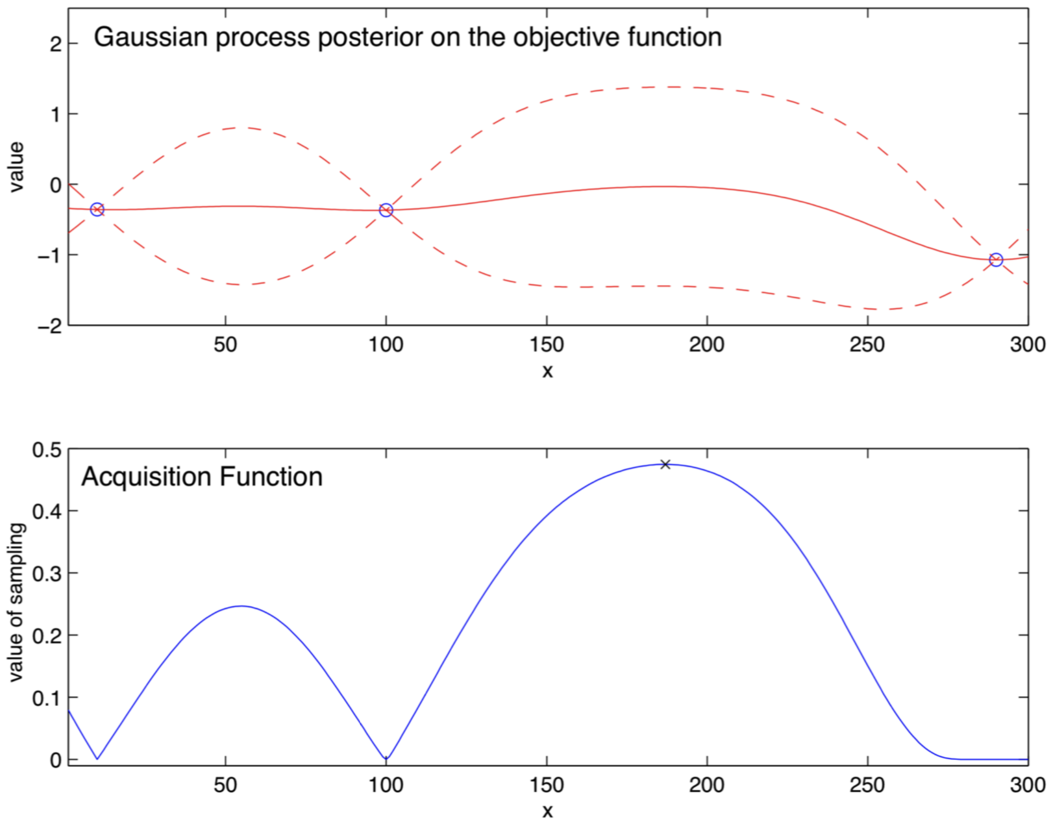


Figure 2.20: Model and acquisition function in Bayesian Optimisation [17]

The statistical model provides a Bayesian posterior probability distribution that describes the possible values of $f(x)$ at a given point x. We use this model as input to the acquisition function, which computes the next point to explore $(x_n)$. After we get this point, we observe the output of this point from the given objective function to obtain new information about the model ($y_n$). Finally, we update our model using this information and iterate again until we run out of budget.

In Figure 2.20, the top panel shows the current model with the blue circles as noise-free observations of the objective function. The model also shows the posterior probability distribution on each f(x) that is normally distributed with mean and variance. The solid red line shows the mean and the dashed red line shows the variance with 95% Bayesian credible interval. The mean is usually approximated to the actual value of the objective function. The variance is used to calculate the next point. As we observe in the bottom panel, the value of the acquisition function correspond to the uncertainty of the model at a given x with the highest value at the most uncertain point. This makes sense because the value of x with highest acquisition function value is chosen as next point to explore and observing a point where we are more uncertain about the objective model tends to be more useful in finding good approximate of the model and thus global optima.

### 2.7.2 Regression of Gaussian Processes

This part aims to evaluate and iteratively update the model with the given information. The first step would be to collect the known values of the points $(x_1...x_n)$ into a vector $[f(x_1)..fx_n()]$. Now we can construct a mean vector and covariance matrix for these values which correspond to a multivariate Gaussian Process with a prior distribution. Therefore we can represent this by the following equation.

$$[f(x_1), ..., f(x_k)] = Normal([\mu(x_1)...\mu(x_k)], [\Sigma(x_1, x_1)...\Sigma(x_k, x_k)])$$

The mean vector is constructed using a mean function $\mu$ for each input $x$ and the covariance is constructed by using the applying the kernel $\Sigma$ to a combination of pair of points $x_i, x_j$. These functions are chosen by the model creator with $\mu$ often have the value as a constant (mostly 0) and a kernel as a function that increases with the value of the smaller value of $|||x_i - x_j||$.

Now once we have this multivariate normal prior distribution, and we wish to infer the value of $f(x)$ at some new point $x$. Using the conditional distribution also called Bayes rule we get:

$$f(x)/[f(x_1), ..., f(x_n)] = Normal(\mu_n(x), \Sigma(x))$$

where (at $\mu = 0$):

$$\mu_n(x) = [\Sigma(x, x_1)..\Sigma(x, x_n)][\Sigma(x_1, x_1)...\Sigma(x_k, x_k)]^{-1}[f(x_1), ..., f(x_n)]$$

$$\Sigma(x) = \Sigma(x, x) - [\Sigma(x, x_1)..\Sigma(x, x_n)][\Sigma(x_1, x_1)...\Sigma(x_k, x_k)]^{-1}[\Sigma(x_1, x)..\Sigma(x_n, x)]$$

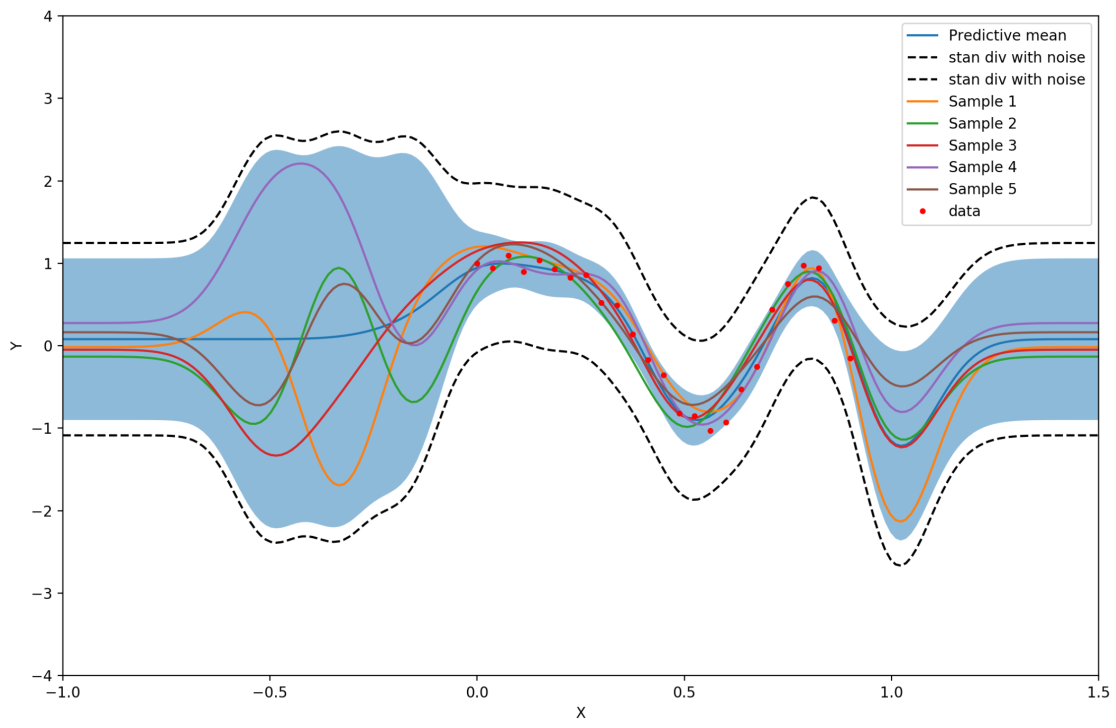


Figure 2.21: Comparing different Gaussian Kernels

Therefore from the above, we can conclude that it is critical to choose right mean functions and kernel. Mean functions as we discussed earlier are constants and the typical value is 0. Kernels should have the property that input points closer are more strongly correlated, i.e. if $||x_i - x_j|| < ||x_i - x_k||$, then $\Sigma(x_i, x_j) > \Sigma(x_i, x_j)$. Most commonly used simple kernel is Gaussian kernel as shown in Figure 2.21 which is often represented by:

$$\Sigma(x_i, x_j) = \alpha * exp(-||x_i - x_j||^2)$$

It is also to be noted that mean function and kernel contain parameters called prior hyper-parameters. There are three approaches to calculate them which are maximum likelihood estimate (MLE), maximum a posteriori (MAP) and sampling the hyper-parameters from their posterior. However, these techniques are only used in very specific applications and are beyond the scope of this project.

### 2.7.3 Acquisition Functions

As we discussed in Section 2.7.1, the acquisition function is used in each loop to find out the next point to evaluate with the given function. There are many established acquisition function; however, we would discuss in detail the most common acquisition functions which are ‘expected improvement’, ‘Upper Confidence Bound’, ‘Probability of Improvement’. They are easy to reason, use and performs well as compared other functions such as ‘gradient’, ‘entropy search’ and ‘predictive entropy’ search which is typically only used in more ‘exotic’ problems.

#### Upper Confidence Bound

Probably the most straightforward function to understand is the upper confidence bound. According to this function, the next point to explore is given by:

$$x_{n+1} = argmax_x(\mu(x) + \kappa * \sigma(x))$$

According to the formulae the next point would be given by the summation of mean and the variance or uncertainty in the prior distribution. This formulae is intuitive because in the maximisation case, we would like to explore the term near the higher value of the known terms (given by $\mu(x)$) to get the local maximum. Further, we would also like to explore the areas with maximum uncertainty to ensure we are obtaining the global maximum. Therefore we add the $\sigma(x)$ term to get higher values with higher uncertainty.

The $\kappa$ term expresses the balance of 'Exploitation vs Exploration' in the function. The higher the value of $\kappa$, the value of the next term would be affected by the uncertainty and the model would be inclined to explore the unknown regions. While having a small value of $\kappa$ makes the contribution of mean in the formulae significant, resulting in model looking for values near the higher terms. Therefore ideally we would require careful calibration of $\kappa$ to explore the region and at the same time to obtain the maximum from the known regions.

**Expected Improvement**

We can illustrate this by an example. Suppose we have completed n iteration and therefore have n observed values. In this case, if we need to decide the maximum, the optimal choice is the previously evaluated point with the most significant observed value or $f* = max_{m \leq n} f(x_m)$.

Now suppose we want to perform one more additional evaluation. We should choose an x such that $f(x) \geq f*$. We can measure this improvement (Em) with $f(x) - f*$ for positive value and 0 otherwise. This can be represented by the following equation.

$$Em(x) := max(0, E(f(x) - f*))$$

We can further calculate the expected value in the above equation using the given mean and variance of the posterior distribution. We can, therefore, get the next point to evaluate by:

$$x_{n+1} = argmax_x Em(x)$$

We should know that certain assumptions have been taken into account in the above equations. These include that we are risk-neutral and do not have an inclination towards any specific values, and we are willing to take the best value (f*) as the final solution if no better alternative is found.

**Probability of Improvement**

Probability of improvement is the first acquisition function designed for Bayesian optimisation. Again in this, we would suppose we have completed n iterations and therefore have n observed values. Now, the maximum or the optimal choice is the previously evaluated point with the most significant observed value or $f* = max_{m \leq n} f(x_m)$.

Now we can get the utility only in the cases when we find a bigger value than the current optimal value. Hence the utility (u) function can be observed as:

$$u(x) = 1 \qquad iff(x) > f*, \quad and$$

$$u(x) = 0 \qquad iff(x) \leq f*$$

The above functions show that we get a unit reward if $f(x)$ turns out to be more than f*. Otherwise, we get no reward. Therefore we can write the probability of Improvement as the cumulative distribution function as shown in the equation:

$$PI(x) = P(f(x) > f*)$$

$$\implies PI(x) = \int_{-\infty}^{f*} N(f; \mu(x), \sigma(x)$$

$$\implies PI(x) = \Phi(\frac{\mu(x) - f*}{\sigma(x)})$$

Where $PI()$ function represent the Probability of Improvement, $N()$ is the normal Bayesian function and $\Phi()$ is cumulative distribution function (CDF) of the standard normal distribution. It is given by the following equations:

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{\frac{-t^2}{2}} dt$$

Finally we can find the next point to evaluate by:

$$x_{n+1} = argmax_x PI(x)$$

Intuitively, we can understand that the probability of improvement first defines a function with a unit reward if we find the better optimal solution and then find input which would maximise the probability of this function.

### 2.7.4 Library

We have opted to use the 'bayesian-optimization' library [65] in python to perform Bayesian optimisation in our algorithm. The main reason is it being open-source and provides with a simple interface to implement and tune the optimisation process. It also offers a wide variety of configurations and parameters which allow to experiment and choose the best algorithm for our optimisation. This library in-turn uses 'Scipy' and 'Scikit-learn' libraries for low-level Bayesian functions.

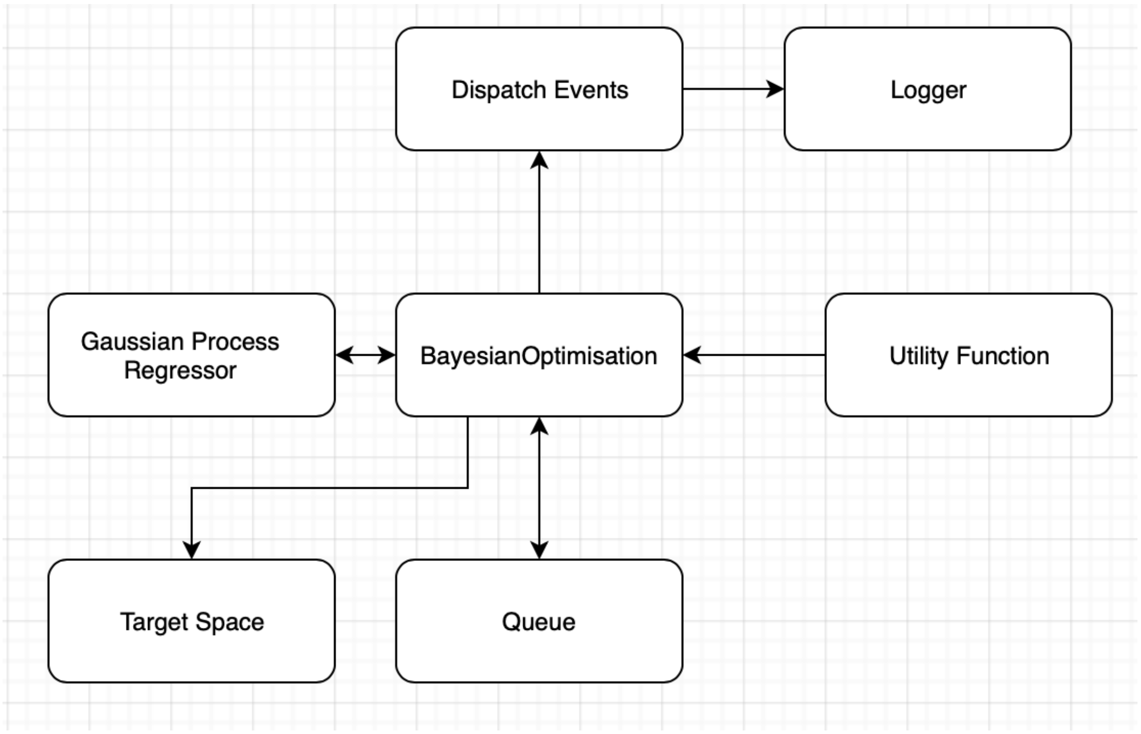


Figure 2.22: Overview structure of the 'bayesian-optimization' library

Figure 2.22 shows the brief structure of the library. The optimiser is defined in the 'Bayesian optimisation' class, which acts as the main class with other parts of the library supporting different helper functions. The optimisation process begins by defining the Gaussian Process Regressor, Target Space and Queue. The Target Space is defined by the optimisation function and the user-given bounds. The specification of the Regressor is defined by the user. We would be using the default kernel for our case, which is Matern[66] with $\nu$ to be 2.5. Further, we have assumed no noise for our experimentation thus we would be using a small value of alpha=1e-6.

After getting the Regressor, we add the initial point to the queue. Then these points are evaluated one by one and added to the Target Space. The Target Space is responsible for interaction with the given user-function. It evaluates the generated points, keeps tracks of the known data and even generates random initial points.

Next, we use this information to update the Reggresor and update our model. This Regressor is then passed to the acquisition function in the Utility Function library to get the next best point to evaluate. This new point is added to the queue and evaluated and updated by the Target Space similar to initial points. The above process is carried in a loop until we run out of iterations.

Periodically after each iteration, the main function dispatches events. These events are picked up by the observer and then passed to the logger for logging. This helps the user to monitor the process and observe the behaviour of the library.

# Chapter 3

# Experimental Methodology

In this chapter, we would discuss in brief the experiment methodology adopted to measure energy usage by machine learning algorithms in Section 3.1, steps are taken to remove background noise in Section 3.1.1, and measures taken to solidify the findings in Section 3.1.2. To calculate the precise energy savings, we would compare the energy consumption of the established algorithms with the energy utilisation of the proposed algorithms. Therefore, we measure the energy consumption of the current established models in Section 3.2 and will discuss and compare the proposed algorithms in further chapters.

## 3.1 Energy Measurement environment

The first step before implementation of any algorithms was to create an ‘Energy Measurement Environment’ that can reliably measure the energy utilised on a given hardware. As we studied in the background, there is no established tool for energy measurement in the industry. The primary reason being that energy measurement varies a lot with different architectures and processors. Moreover, as we discussed in Section 2.5.1, in some applications such as DNN’s most of the CPU cycles and energy is consumed in the data movement rather than processing of data. Thus the energy consumption of RAM and disk can be significant. Therefore, this makes it harder for any tool to give holistic energy consumption for all possible architectures and energy sources.

```python
def start_measurement(name):
    global p
    #Start subprocess to record time and energy taken
    p = subprocess.Popen(["rm", "-f", "./_s-tui" + name + ".log" ], shell=False)
    #print "Starting s-tui @", datetime.now().strftime('%Y-%d-%m-%H-%M-%S')
    f = open("output.txt", "w")
    p = subprocess.Popen(["s-tui", "-d", "--debug-file", "./_s-tui" + name + ".log" ], stdout=f, stdin=subprocess.PIPE, shell=False)
    return
```

Listing 3.1: Code snippet showing the function to start energy measurement

We would primarily use s-tui for our energy measurements. The main reason being that it collects information directly from RAPL interface and psutil library, which are both very widely used and reliable sources of system information. Moreover, it is the closest universal tool which supports Linux OS with Intel processors (Intel processor generation from 2 to 7).

Listing 3.1 shows the Python implementation for the start of energy measurement. We use the subprocess library to start s-tui process in the background. The measurements are recorded in the log file passed (‘_s-tui + name’ in this case). Further, we PIPE the output of the file into a separate output.txt file to get a cleaner output from the program.

```
2019-05-10 09:28:46,091 [update()] [INFO ]  Utilization recorded 77.8
2019-05-10 09:28:46,092 [get_power_usage()] [INFO ]  current 1.49292993602e+11 last 1.49292107679e+11
2019-05-10 09:28:46,092 [get_power_usage()] [INFO ]  Joule_Used 0.885923 seconds_passed 0.0110728740692
2019-05-10 09:28:46,092 [get_power_usage()] [INFO ]  Power reading elapsed
2019-05-10 09:28:46,092 [get_power_usage()] [INFO ]  Max power updated 81.0
2019-05-10 09:28:46,092 [update_displayed_graph_data()] [INFO ]  Reading 4399
2019-05-10 09:28:46,092 [update_displayed_graph_data()] [INFO ]  Reading 77.8
2019-05-10 09:28:46,092 [update_displayed_graph_data()] [INFO ]  Reading 76.0
2019-05-10 09:28:46,093 [update_displayed_graph_data()] [INFO ]  Reading 80.0084056282
2019-05-10 09:28:46,094 [update()] [INFO ]  Utilization recorded 66.7
2019-05-10 09:28:46,095 [get_power_usage()] [INFO ]  current 1.49293268626e+11 last 1.49292993602e+11
2019-05-10 09:28:46,095 [get_power_usage()] [INFO ]  Joule_Used 0.275024 seconds_passed 0.00315713882446
2019-05-10 09:28:46,095 [get_power_usage()] [INFO ]  Power reading elapsed
2019-05-10 09:28:46,095 [get_power_usage()] [INFO ]  Max power updated 88.0
```

Listing 3.2: An example log file for the energy measurement

Listing 3.2 shows an example log file. We can observe different readings from a graph such as Temperature, Frequency, Utilisation etc. It also records the total power utilisation in Joules and seconds passed.

```python
def stop_measurement(name):
    global p
    # Stop the record of time and energy
    p.stdin.write('q')
    #print "Killing s-tui @", datetime.now().strftime('%Y-%d-%m-%H-%M-%S')

    #Calculate total energy used
    logbuffer = open("./_s-tui" + name + ".log", 'r').read()
    pattern = re.compile("Joule_Used\s+[0-9]+\.[0-9]+\sseconds_passed\s+[0-9]+\.[0-9]+")
    matches = re.findall(pattern, logbuffer)

    energy = []
    times = []

    for match in matches:
        val = float(match.split(' ')[1].strip())
        energy.append(val)
        val = float(match.split(' ')[3].strip())
        times.append(val)
    return (sum(energy), sum(times))
```

Listing 3.3: Code snippet showing the function to stop energy measurement

We stop the measurements by killing the background process that records the energy. Then we parse the file to calculate the total energy used and time passed, as shown in Listing 3.3. Thus we get the total energy used and the total time the program ran.

All the experiments are run in an isolated machine with 8 core cpu with 2 thread(s) per core. The CPU model is Intel(R) Core(TM) i7-7700K CPU @ 4.20GHz with CPU max frequency at 4500 MHZ and min frequency at 800 MHZ. The machine has L1d and L1i cache of 32K each. It also has L2 and L3 cache of 256K and 8192K respectively. The system has a total 16Gb of memory with SSD for storage.

### 3.1.1 Accounting for background noise

Background noise includes other user programs and system applications running. These can contribute significantly to the total energy consumption of the system. Therefore we account for this by running sleep for 5 seconds and record the energy consumption by s-tui tool.

Once we get this reading, we calculate the average reading per second. This gives the energy consumption by the background processes per second. Calculating per second allows us to calculate the expected energy consumption for the application as we know the run-time of the application. This is then subtracted from the total reading of the application energy to get our final energy reading.

We take this background reading each time just before taking the application reading. We, therefore, have an underline assumption that the background energy consumption does not vary substantially during the evaluation phase as compared to background reading phase. This could be a reasonable assumption because the evaluation phase typically lasts for a few seconds and therefore, the probability of significant change of average background energy consumption in a few seconds just after the reading is small.

### 3.1.2 Verify results

Although we use s-tui as our primary tool for energy consumption, we further verify our results using the other energy measurements tools; PowerKap and Power Gadget. These tools are briefly explained in Section 2.3.

The main advantage of using PowerKap over s-tui is that it provides the entire RAPL reading as output. This allows to exactly measure the energy consumption by the CPU cores and RAM. Having this information could be useful as we can ignore the background noise if we dedicate a CPU core to the user program. In such case we can only account for the energy consumption by this core and noise is only limited to energy consumption in L3 cache and memory.

We use Intel Power Gadget as a visualisation tool. This tool plots the live and historical energy consumption, hence making it easier to analyse energy consumption and quickly observe any variation in energy consumption either during the program or due to background noise.

Finally, measuring the results multiple times with different tools also helps to solidify our readings. This also allows us to account for the unlikely events where we observe a significant variance in the background noise due to system processes.

## 3.2 Measuring energy consumption of state-of-the-art solutions

Now, after establishing our experimental methodology, we would use this to measure the energy consumption of the state-of-the-art solutions in this field. We have taken two implementations as benchmarks for our project; Keras library implementation [67] for Cifar10 and MNIST and cascading models [12] (Scalable classifiers in Section 2.5.3).

Keras library implementation (also known as KerasNet) is one of the most popular CNN model used for Cifar10 and MNIST and is widely used for comparison in studies [68] [69] [70]. It's a single CNN with a few layers. The main parameters of the network are given in Table 3.1.

| **Hyper-Parameters** | **Value** | **Explanation** |
|---|---|---|
| conv1_features | 32 | Dimensionality of output space of convolution layer 1 |
| conv1_kernel | 3 | Height & Width of 2D convolution window of layer 1 |
| conv2_features | 32 | Dimensionality of output space of convolution layer 2 |
| conv2_kernel | 3 | Height & Width of 2D convolution window of layer 2 |
| conv3_features | 64 | Dimensionality of output space of convolution layer 3 |
| conv3_kernel | 3 | Height & Width of 2D convolution window of layer 3 |
| conv4_features | 64 | Dimensionality of output space of convolution layer 4 |
| conv4_kernel | 3 | Height & Width of 2D convolution window of layer 4 |
| full1_features | 512 | Dimensionality of output space of fully connected layer |
| Activation functions | relu,softmax | Defines the output of that node |

Table 3.1: Table explaining hyper-parameters of Keras CNN

This implementation is simple as compared to more recent Neural Networks and thus provides 74% accuracy with 30 epochs and about 78% accuracy with 100 epochs. Being a simple Neural Network, it consumes much less energy and therefore poses a challenge for our project to optimise for energy-efficiency. The goal of the project is not to create the best accuracy network but develop techniques to reduce the energy consumption while providing a similar accuracy of a given network. Therefore, an advantage of choosing such a simple network is that it has shorter training time providing more time to experiment.

We have taken multiple readings (5 in total) of the Keras example model for the MNIST and Cifar10 datasets. Figure 3.1 and Figure 3.2 show the accuracy and total energy consumed by the model for MNIST dataset, respectively. We obtain average application energy (different of total energy and background noise) of 114 Joules and an average accuracy of 99%.

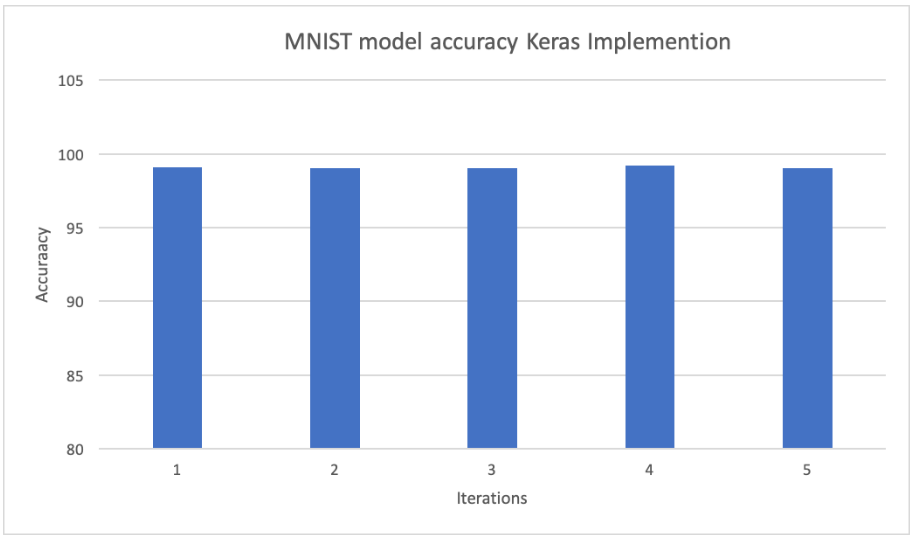


Figure 3.1: Multiple reading of accuracy of MNIST Keras model

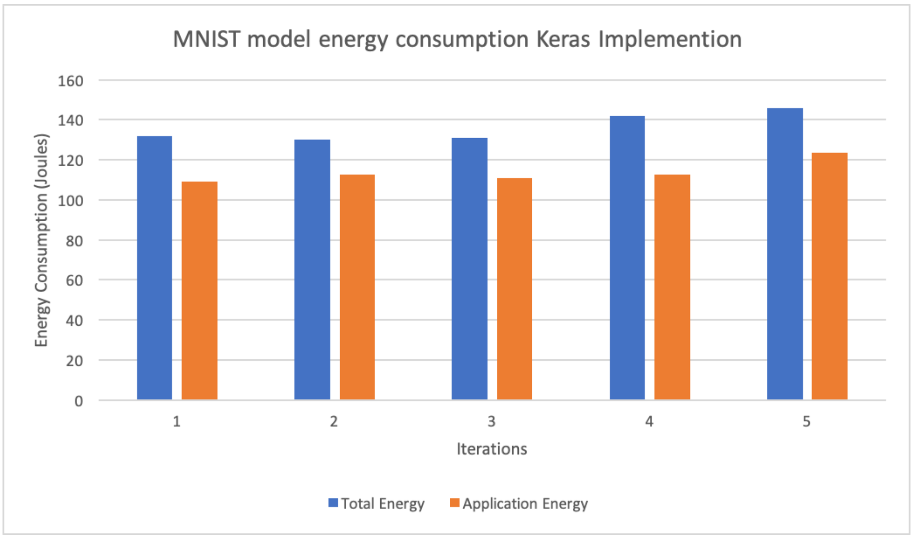


Figure 3.2: Multiple reading of energy consumption by MNIST Keras model

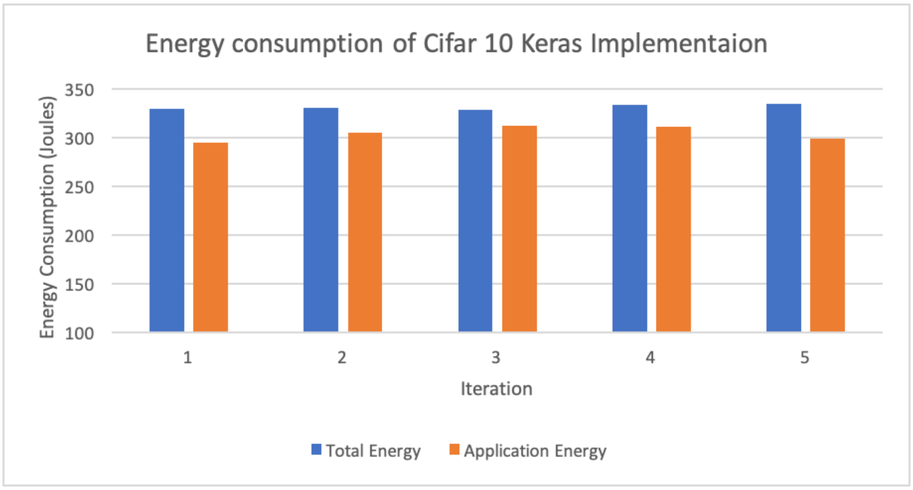


Figure 3.3: Multiple reading of energy consumption by Cifar10 Keras model

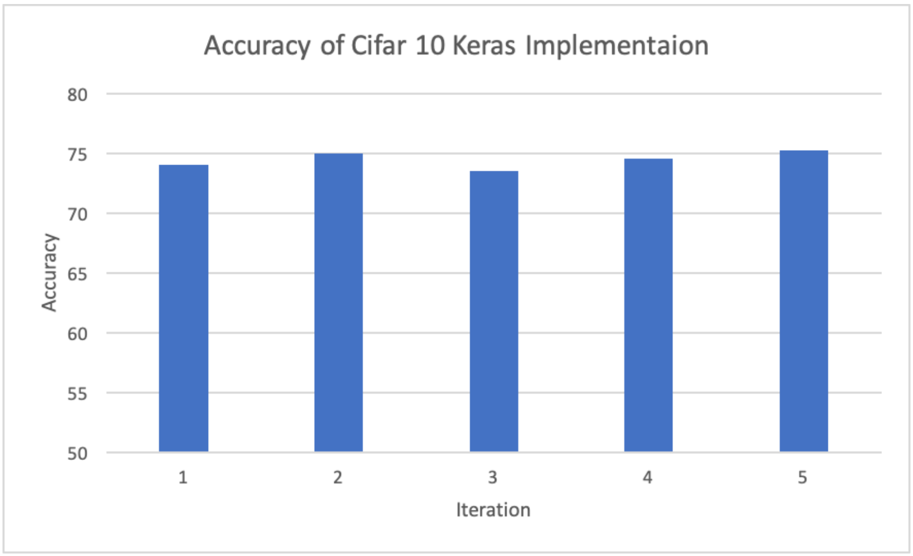


Figure 3.4: Multiple reading of Accuracy of Cifar10 Keras model

Figure 3.3 and Figure 3.4 show the energy consumption and accuracy of the Keras model for the Cifar10 dataset. Similar to MNIST, we observe the total energy consumed and the application energy. In this case, we observe average energy of 304 Joules and average accuracy of 74%.

Scalable classifier [12] is also one of the recent and most popular technique by Microsoft in Cascading networks. The basic idea being to train two different networks; one low-cost network for 'easy' inputs and other more accurate higher cost network for 'difficult' inputs. We will evaluate and compare this technique in this section.

For this project, we have defined the low-cost network or simple network for easier input as *Initial model* and the more accurate higher cost network for difficult inputs as *Final model*.

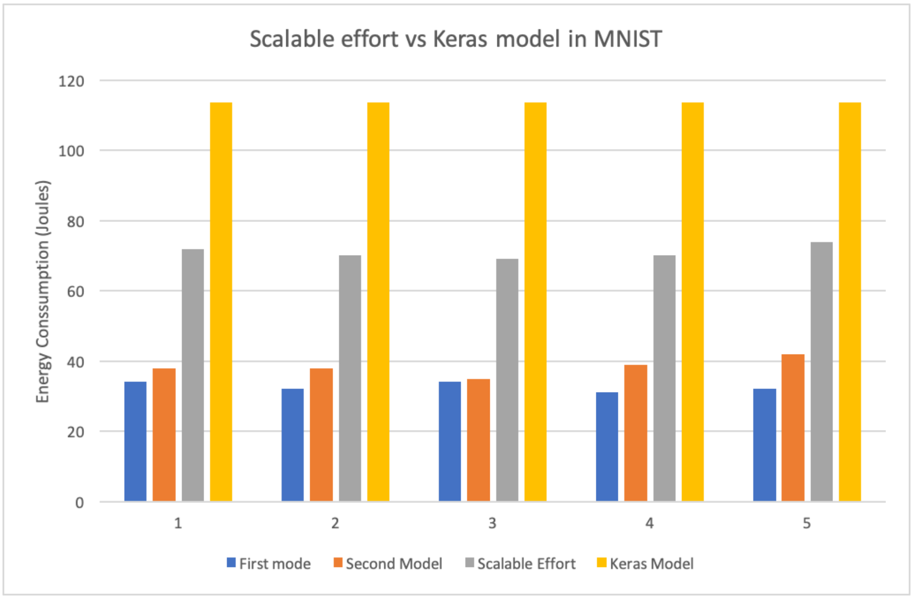


Figure 3.5: Energy consumption by MNIST Keras model and Scalable effort model

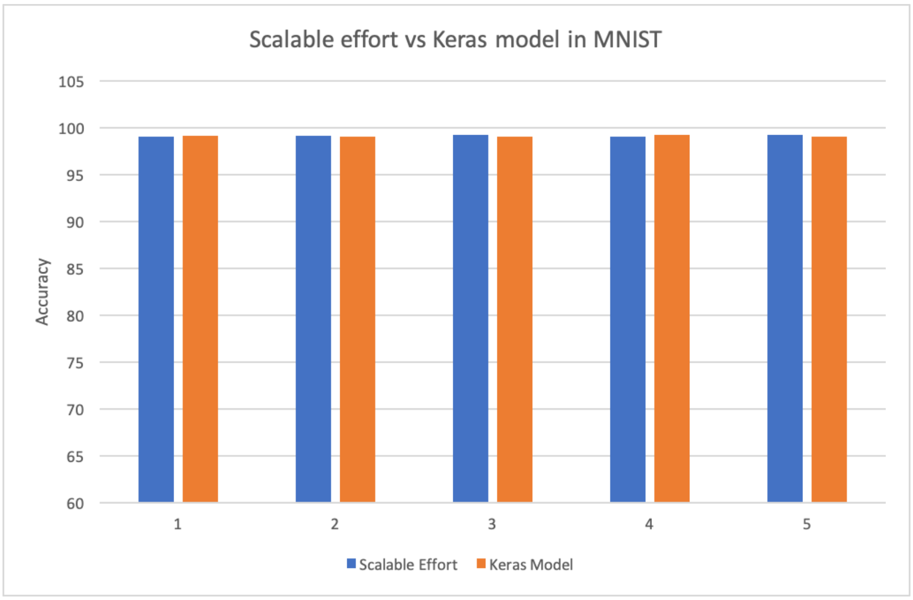


Figure 3.6: Accuracy of MNIST Keras model and Scalable effort model

We have implemented the scalable classifier [12] (Scalable effort model) and compared the result with the Keras example, as shown in Figure 3.5. The net energy consumed is calculated as the summation of the energy of the 2 models used. We observe an average of 71 Joules, which is a significant (37%) improvement as compared to the one classifier model (Keras Model). While at the same time, we observe almost no change in the accuracy of the network, in Figure 3.6.

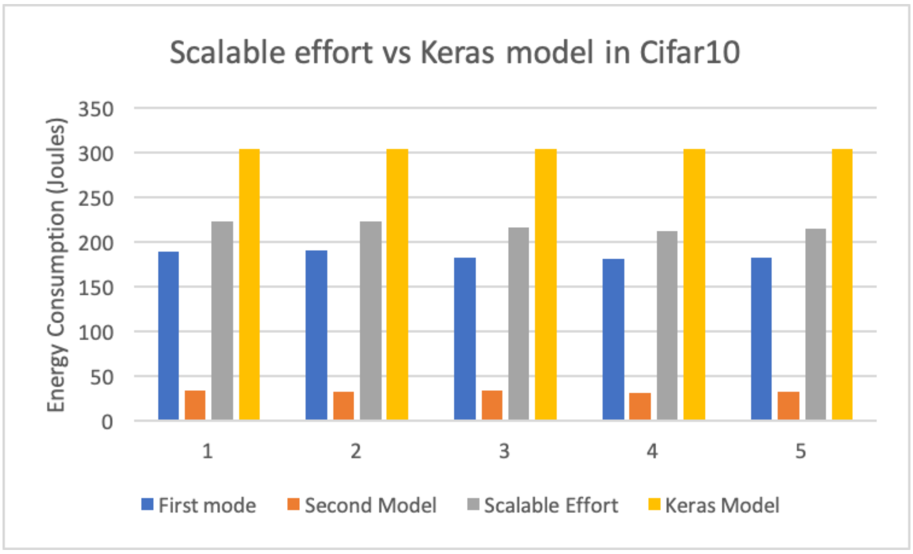


Figure 3.7: Multiple reading of energy consumption by Cifar10 Keras model and established Scalable effort model

Similarly, in Figure 3.7, the total scalable model energy is the summation of the energy of the two models used. We get an average of 218 Joules for the Scalable model. If we compare it with the Keras model with energy consumption of 304 Joules, we get 28% reduction in energy consumption. Another observation is, due to the more complex input space, the Initial model (first model) takes most of the energy of the Scalable model energy. In our observations, the first model takes 5× energy as compared to the second model. Figure 3.7 shows that in the case of accuracy, the scalable model outperforms the Keras model by about 1.5%.

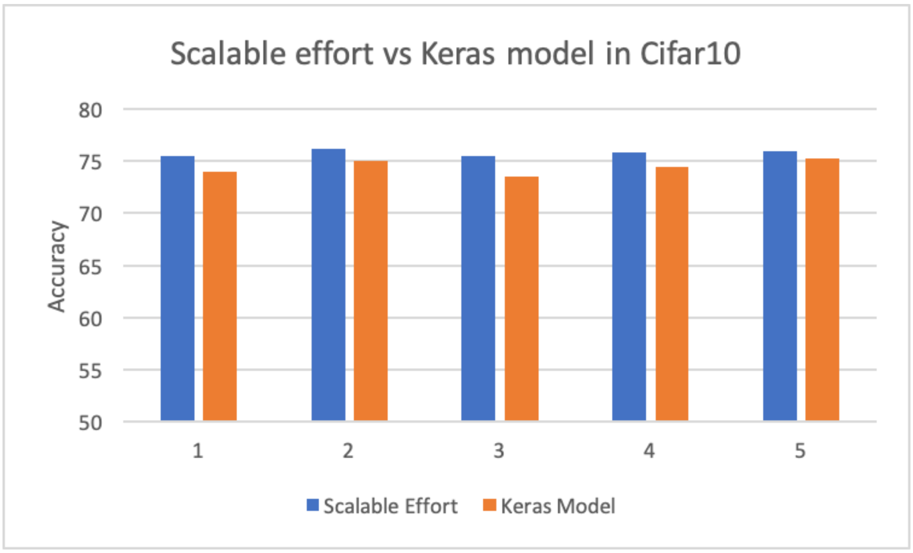


Figure 3.8: Multiple reading of accuracy by Cifar10 Keras model and established Scalable effort model

To conclude, we first developed an Energy Measurement environment to reliably measure the energy consumption of given neural networks. Then we used this environment to measure the accuracy and energy consumption of two popular network approaches; Keras model and Scalable effort model. We observed that Scalable effort model outperforms the Keras model in the accuracy and is still able to save a substantial amount of energy (37% and 28%). This proves the effectiveness of the cascading model technique. Now, we will explore methods to improve this Scalable effort model further.

Chapter 4

# Using lower dimensional input data

In this chapter, we would discuss different proposed optimisation techniques to improve the current state-of-the-art cascading models by reducing the input 'dimensionality'. In Section 4.1, we discuss different techniques to extract Semantic data such as Colour, Texture, Edges, etc. and methods to Systematically reduce the dimension of the input space. Then in Section 4.2, we discuss our proposed algorithm to arrange the Initial models such as to gain maximum accuracy. Finally, in Section 4.3, we measure and evaluate the results corresponding to our algorithm.

One of the main bottle-neck for the Scalable effort techniques in Section 3.2 was the complexity of early models. These early models (Initial models) took up a large amount of energy for their predictions. This was observed due to large input layer due to the complexity and size of input images. For example, in Cifar-10, each input image had the size of 32*32*3, which would require 3072 input values for each image.

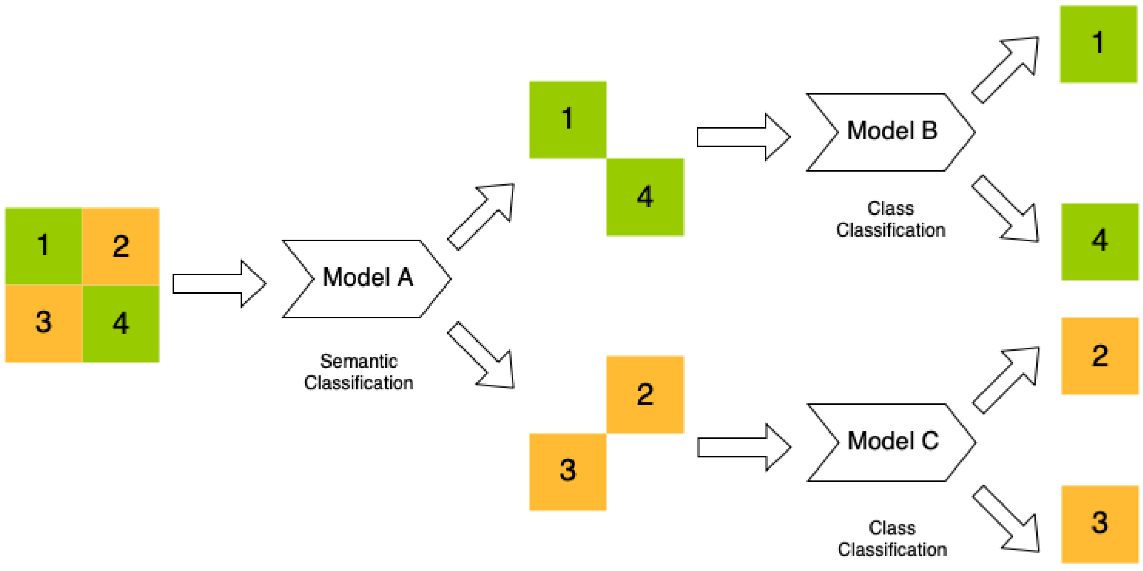


Figure 4.1: Illustration for the process to pre-classify classes based on the semantic data

In the real world, we can extract a large amount of data from images; this includes broad characteristics like colour, texture, edges etc. Thus, we can reduce and simplify the given dataset into these characteristic (semantic) data. This allows to significantly reduce the input complexity, thus allowing for much simpler models. We observe this effect in Section 3.2 when comparing total energy of cascading model with a single model in MNIST and Cifar10 databases.

The basic idea can be understood with Figure 4.1, in this case, we classify the data for four classes into two groups (1&4 as a group, 2&3 as another group) first with the semantic classification as done by Model A. This classification should be simple due to the small input size. Once small groups with similar semantic characteristics are formed, we can further classify them to obtain individual classes using the full input size of the image. This is shown with Model B and C, since these models have to only classify on a few classes they could also be much simpler than the initial complex model.

If we compare the model in Figure 4.1 to the traditional cascading model, we have been able to split the Initial model further into simpler models. The net energy savings would be the difference of energy consumed by the given initial complex model and summation of energy consumed by these simpler models in a given path. It is to be noted that the effectiveness of this technique is reliant on the accuracy of the semantic model, which in-turn depends on the chosen semantics and the given dataset. Further, we also need to develop an algorithm to arrange and select these models effectively.

## 4.1 Extraction of semantic data

The first step for this type of classification would be to extract the semantic data, which includes colour, texture, edges and corners. We would have to use many transformation, detection and filtering on the raw images to extract this data. It is to be noted that these techniques also require energy which should be factored in to calculate the overall energy benefit for the algorithm. In the next few sections, we explain each attribute of data extraction.

### 4.1.1 Colour extraction using Hue Saturation Value (HSV) transformation

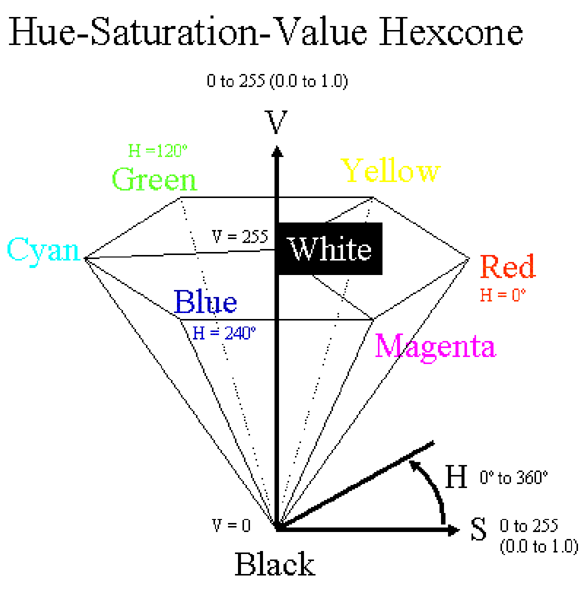


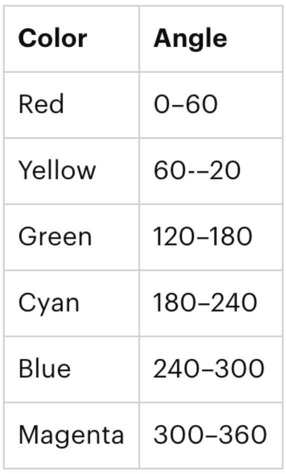
Figure 4.2: Hexcone describing different components of HSV [18]

HSV (hue, saturation, value) are alternative representations of the CMYK (cyan, magenta, yellow, key) and RGB (Red Green Blue) colour model. These are all different schemes used to define basic colour combinations that humans can see around in the various media.

However, unlike RGB and CMYK, which define the primary colour components, HSV is defined in a way that is similar to how humans perceive colour [19].

There are three main parts of the HSV transformation.

**Hue**: Hue is described by the dominant wavelength or a dimension of colour we readily experience when we look at the colour. It is parallel to the full saturation determined by the ratio in the RGB colour scheme [20]. The colours in the hue model can be expressed as a number from 0 to 360 degrees, as shown in Figure 4.3. In the graphical model, as expressed in Figure 4.2, Hue is shown by the direction of the vector in x-y plane, or the degree from X-axis.



| Color | Angle |
|---|---|
| Red | 0–60 |
| Yellow | 60--20 |
| Green | 120–180 |
| Cyan | 180–240 |
| Blue | 240–300 |
| Magenta | 300–360 |

Figure 4.3: Colours with the corresponding degree value in hue component of HSV [19]

**Saturation**: Saturation defines the ‘intensity’ or ‘chroma’. It refers to the dominance of the given Hue (colour) [20]. Another method to interpret this is the amount of grey in the colour. Saturation is often expressed in a range from just 0-1 or in 0-255. In the graphical model, as shown in Figure 4.2, the saturation is described by the length of x-axis. Figure 4.4 shows the top-down view of the model; in this case, we view the saturation level as 100% with hue angle towards yellow colour.

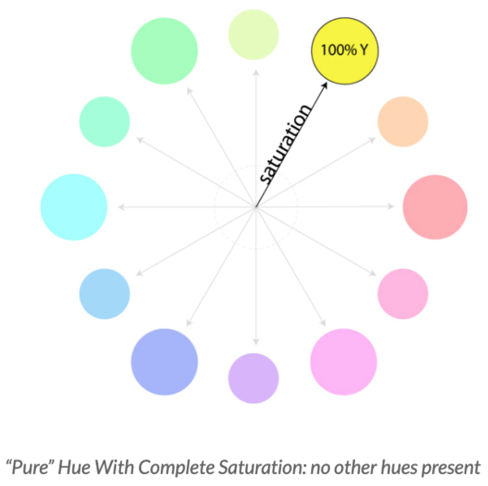


Figure 4.4: Top down view of the HSV graphic model [20]

**Value**: This value represents the intensity or the dimension of brightness or darkness. It is often represented in the range of value from 0 to 1, where 0 is completely black, and 1 is the brightest and reveals the most colour. Figure 4.5, shows the effect of the value. As we move upwards along the z-axis, the colours become brighter, and as we move downwards, the colour becomes darker.

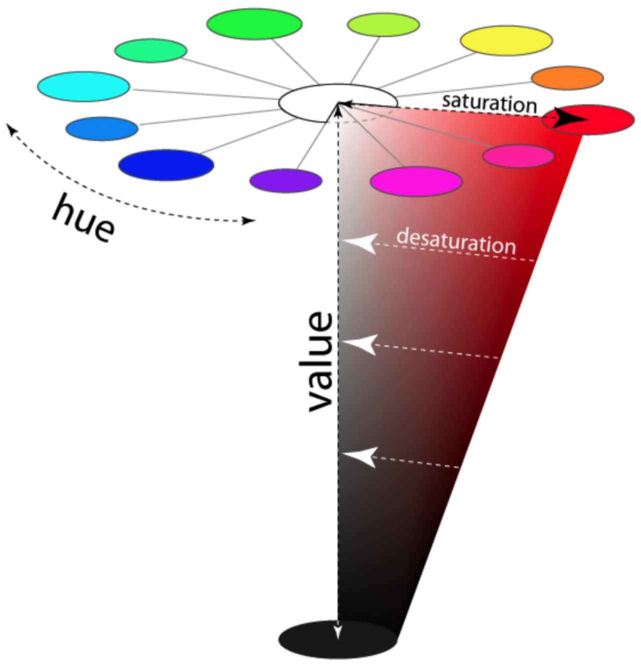


Figure 4.5: Show Variation of colour with change in 'value' [20]

The process begins by defining a filter with an acceptable range. This range is defined by an upper and lower bound of Hue, Saturation and Value. This filter now is used to mask all the pixels which not belong to this specific range. For our implementation, we are varying colours or Hue. Thus a typical mask in our implementation can be ((30,0,0), (90,255,255)). This mask accepts all the tuples with Hue between 30 and 90, and Saturation and Value between 0 and 255.

Thus by introducing this mask, we ignore all the pixels which belong beyond to the defined range. Suppose we create six non-overlapping filters, in this case, we can on average get 16.6% of the original pixels. This thus reduces the input size and help us create much smaller models.

### 4.1.2 Texture extraction using Gabor filtering

Gabor filters are orientation-sensitive filters, used for texture analysis. The response of these filters is created by multiplying a Gaussian envelope function with a complex oscillation. Put simply; they are mathematical structures which take care of different shapes, sizes and smoothness levels in the image. If a Gabor filter is oriented in a given direction, it will produce a strong response if the target image also corresponds to the same direction [71].

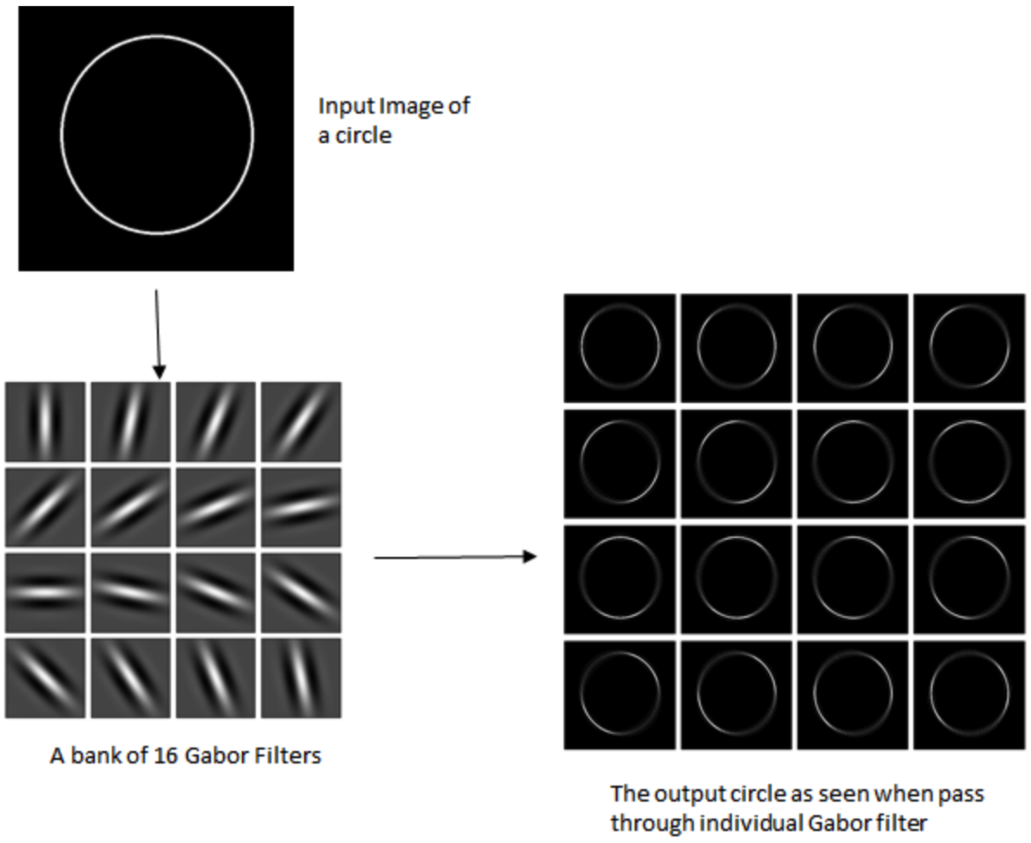

Input Image of
a circle
A bank of 16 Gabor Filters
The output circle as seen when pass
through individual Gabor filter

Figure 4.6: Process to apply Gabor filters to a given image [21]

In Figure 4.6, we take the input image of a simple circle and pass it through 16 different filters. Now, this bank of 16 Gabor filters at an orientation of 11.250 (i.e. if the first filter is at 00, then the second will be at 11.250, the third will be at 22.50, and so on [21]). Thus in the output resulting images, we can observe only the parts oriented with the corresponding filters are enhanced.

For our application, we have applied compared eight different Gabor filters with different thetas. We have kept the values of lambda (wavelength governs the width of the strips of Gabor function), Gamma (the height of the Gabor function), sigma (controls the overall size) as constants.

Firstly, since the input for Gabor filtering is grey-scale, we reduce the information to 33% as we need one channel to represent the image. Further Gabor filtering also perform the non-linear reduction in the size of the image. However, we can expect a 50% reduction in pixels as the pixels which are aligned perpendicular to the filter should be eradicated.

### 4.1.3 Edges detection using Sobel detectors and Laplacian filters

We can define an edge in an image as a collection of pixels whose grey value is a step up or a roof change when compared to pixel values surrounding it. It can also refer to the brightness of the local area when compared to the rest of the area. Edge detection is mainly the measurement, detection and location of the changes in image grey. Image edge is the essential features of the image and therefore, can prove as useful feature extraction [72].

The basic idea of edge detection relies on the above definition. It sets a threshold of 'pixel edge strength'. When we compare the pixels in the local area with neighbouring areas, if the difference is above this threshold, it is considered as an edge.

Using the Sobel edge detector has two main advantages. Firstly, since the introduction of the average factor, it can smooth out any random noise in the image. Secondly, since it takes differential of both two rows and column at once, it leads to enhancement of edge on both sides while filtering giving a thicker and brighter edge as output [72]. An example of Sobel detection is shown in Figure 4.7.

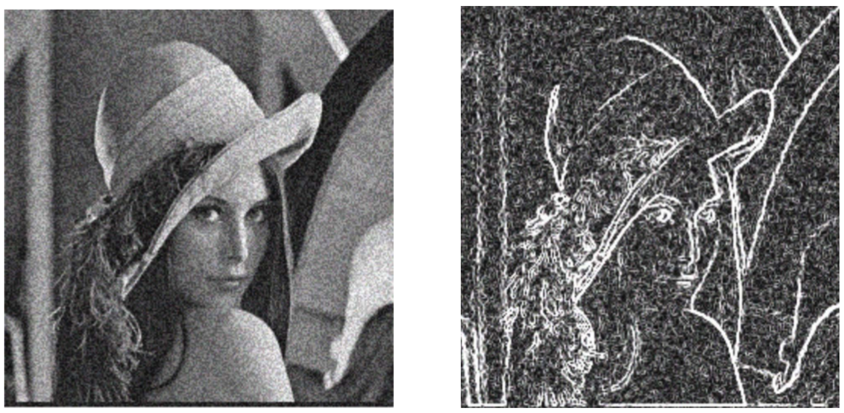

(a) original image (b) After Sobel detector

Figure 4.7: Show image before and after Sobel detector

We are using Sobel in 3 possible combinations which are single order derivation for the x-axis, y-axis and both x and y axis. We have kept the kernel size (the window size) as constant for all filters.

Further, we have also used Laplacian filters. The main advantage Laplacian has over the Sobel detector is that it does not provide information about edge direction as it uses isotropic operator (equal weights in all directions). Therefore we don't miss any pixel information which is not oriented in a precise manner. Further, it is computationally cheaper to implement than the Sobel as it requires only one mask [73]. An example of Laplacian filters is shown in Figure 4.8.

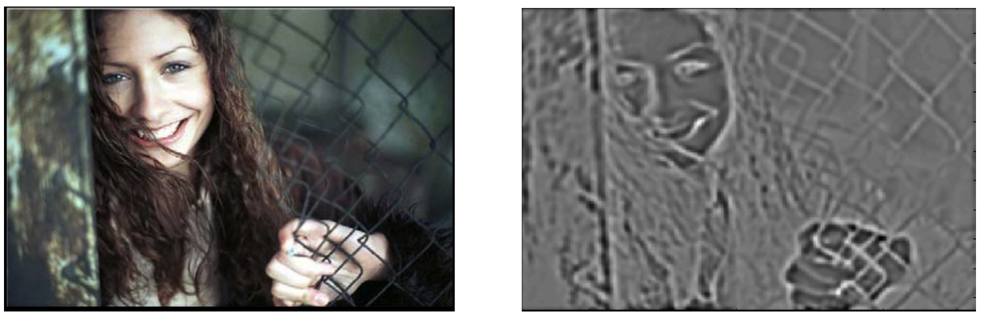

(a) original image (b) After Laplacian filter

Figure 4.8: Show image before and after Laplacian filter

Similar to the Gabor filtering, by grey-scale transformation, we reduce the information to 33% as we need one channel to represent the image. It would be hard to quantify the reduced pixel information, as edges are dependent on the intensity and orientation of the input image pixels. However, if we observe the results from Figure 4.8 and Figure 4.7, we can conclude that only a few percentages of pixels retain their value. Thus we can reduce the dimensionality of the input data.

### 4.1.4 Corners extraction using Harris Corner Detection

A corner is defined as a collection of pixels (which represent a local area) which has a significant change in intensity when compared to other pixels in all directions. Most algorithms define a window size and compare the change of intensity of each pixel with the rest of the pixels in the window to detect if corners fit their definition.

Harris Corner Detector was introduced in ‘A Combined Corner and Edge Detector’ in 1988 [74] by Chris Harris and Mike Stephens. It finds the difference in intensity for a displacement of (u,v) in all directions. This can be expressed by the following equation.

$$E(u,v) = \sum_{x,y} w(x,y) \quad [\underbrace{I(x+u, y+v)}_{\text{Shifted Intensity}} - \underbrace{I(x,y)}_{\text{Intensity}}]^2$$

where,
$I()$: Intensity measurement function
$w()$: window function which gives weights to pixels underneath

We can detect a corner if we can maximise the $E()$ function, i.e. maximise the intensity of point (u,v) with respect to all points represented by $x, y$. An example of this technique detection can be seen in Figure 4.9. In this Figure, we have shown the corners information overlapping the initial image to get better visualisation. After the extraction, we would only send the corner information which would be far less pixel information as compared to the original image – thus providing lower-dimensional data to represent the image.

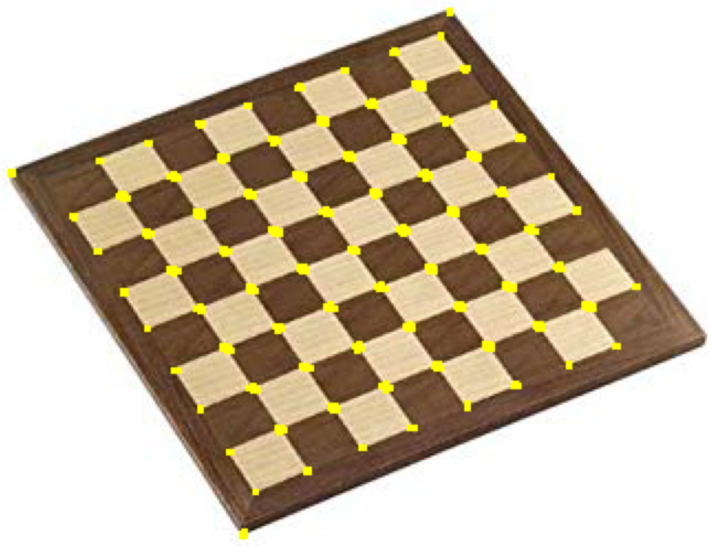


Figure 4.9: An example after applying Harris Corner to a given image

### 4.1.5 Systematic dimension reduction

Until now, we have discussed only semantic techniques to extract information from the input. However, there are also popular techniques, as discussed in Section 2.6.1, to perform dimensional reduction on given input data directly. Given our labelled datasets we would be using LDA over PCA as we want to compress the data such that we also group the classes. We have explained the difference between the two in Section 2.6.1.

The first step would be feature scaling. We have used Standard Scalar to scale all the features. It is performed by removing the mean and scaling to unit variance from each feature. It is represented by the following formulae.

$$\hat{x_i} = \frac{x_i - \mu}{\sigma}$$

where,

$x_i$: The input feature

$\mu$: The feature mean

$\sigma$: The feature variance

Next, we perform Linear Discriminant Analysis (LDA) as described in Section 2.6.1. In our case, we would need to choose the output dimension. Having more dimensions would increase input size; thus, the Neural Network should perform better due to access to more information. Note we are limited to $n-1$ output dimensions where n is total output classes. This is primarily because we need only $n-1$ dimensional space to maximise the separation between classes. For example,, if we have a 2-dimensional plane with 3 groups of data. Then we only need one line that passes through the mean of the classes to represent that data. This phenomenon is explained in Figure 4.10.

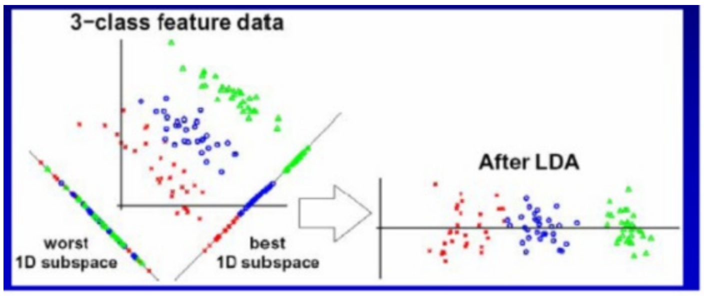


Figure 4.10: Performing LDA on 2 dimensional plane [22]

After performing LDA, we provide this compressed (lower-dimensional) data to classify in a Neural Network. However, unlike the semantic approaches, we do not need to use CNN as the data is no longer in the image format but is represented by the new dimensions. Using a simpler DNN with few features can result in a reduction in energy consumption as compared to using CNN [36].

We also observe a significant reduction in the input size of the data by using LDA. As an example, in Cifar10 dataset, the input dimension would be 32*32*3 (3072), while after performing LDA, it would only be 9 at maximum.

## 4.2 Proposed Algorithm

The next step would be to arrange these low-cost (in terms of energy consumption) classifiers obtained by training lower-dimensional data, to maximise the energy savings without affecting the accuracy.

### 4.2.1 Initial Design

After the extraction of semantic data, we can reduce the input complexity for the simple model in Scalable effort classifiers from Section 2.5.3. This allows us to train cheap (in terms of energy consumption) Initial model to filter out 'easy examples'. Therefore by replacing the original simple model, we get the following design, as shown in Figure 4.11.

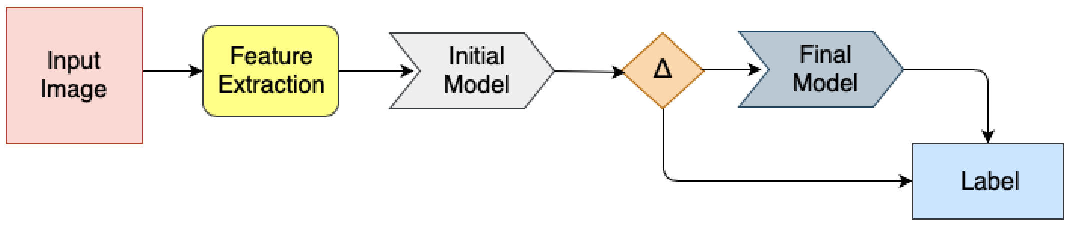


Figure 4.11: Initial Design of the algorithm

In Figure 4.11, first, the required features are extracted from the input image (test image). This step converts the input space from (32*32*3) to a lower-dimensional space. Then this lower-dimensional input data is given to the Initial Model (simple model) for classification. We compare the confidence interval of the result of this Initial model with our confidence threshold (Δ). If we confidence interval of the output is greater than the threshold, we accept the classification and assign the Label. Otherwise, the image is passed to the Final model. This is typically a high accuracy and high energy consumption state-of-the-art model. Therefore after the classification from this model, we accept the result as the final Label.

Choosing the right value of Δ is critical to managing accuracy vs energy consumption. If we choose a very high value of Δ then more and more test images would be passed to the Final model yielding better accuracy with a high amount of energy consumption. While, if we choose a too low value of Δ, few images would be processed by the Final model giving high energy savings with the accuracy as cost. In our experiments, we found a value between 0.6 and 0.8 to give a good accuracy vs energy consumption balance.

$$E_{saving} = E_{final} - (E_{initial} + (1 - f)E_{final})$$

subject to,

$$E_{initial} + (1 - f)E_{final} < E_{Final}$$

$E_{initial}$: Energy consumption by Initial model
$E_{final}$: Energy consumption by Final model
$E_{saving}$: Net energy savings
$f$: Fraction of inputs that are filtered out

The above equations describe the condition and the total energy savings when applying this algorithm.

#### Multiple Initial Models

We can have more than one Initial model to filter out the images further before they reach the Final Model. This can be acceptable as long as the images filter out are large enough so that they can save net energy. The following equations can express this.

$$\begin{aligned} E_{saving} = E_{final} - E_{initial1} - (1-f_1)E_{initial2} - (1-f_1)(1-f_2)E_{initial3} \\ ... - ((1-f_1)(1-f_2)..(1-fn))E_{final} \end{aligned} \tag{4.1}$$

subject to,

$$E_{initial1} + (1-f_1)E_{initial2} \quad ... + ((1-f_1)(1-f_2)..(1-fn))E_{final} < E_{Final}$$

$E_{initiali}$: Energy consumption by ith initial model
$E_{final}$: Energy consumption by Final model
$E_{saving}$: Net energy savings
$f_i$: Fraction of inputs that are filtered out by ith model

Ideally, we can add Initial models until either we surpass the Energy consumption by the Final model or reach energy-saving goals set by the user. Figure 4.12 shows the visualisation of such an algorithm.

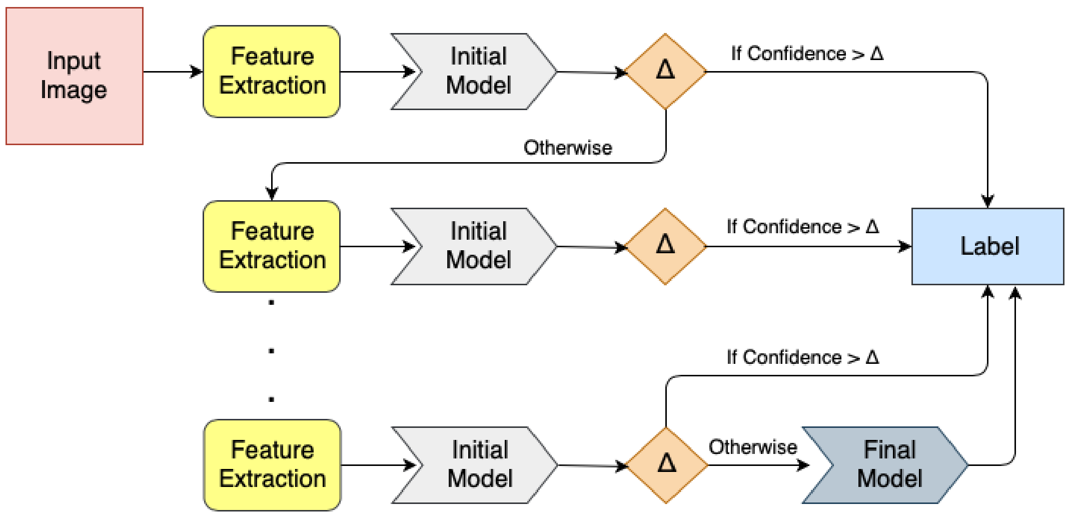


Figure 4.12: Initial Design of the algorithm with multiple Initial models

The above design is similar to before, but in this case, we observe that if the confidence bound is less than the confidence threshold (Δ), we move to the next Initial model. This is done until we run out of all the models as defined by the constants in the equation. In this case, the test image is passed to the Final model for labelling.

### 4.2.2 Choosing feature(s) for Initial design

It is evident that the performance of the Initial model will largely depend upon the choice of features extracted and the corresponding Initial model trained to classify using it. There are two main parameters while choosing an Initial model; energy consumption and accuracy.

To quantify the net effect of energy usage, we need to consider the energy consumption during the feature extraction and model evaluation. Some features such as corners take a lot of energy to extract the dimensions but result in a small Initial model due to small input space (as shown in Section 4.3). While others like colour extraction are cheaper with feature extraction but have a relatively large Initial model. Therefore to compare models, we take the total sum of the energy used during feature extraction and model evaluation.

The accuracy of the model can be simply measured as the percentage of correct outputs. Therefore to account for both the factors we take the approximation that the fraction of inputs filtered out can be the same as the accuracy of the Neural Network. Hence we get the optimal feature with maximum energy saving described by the following equation.

$$E_{saving} = acc * E_{final} - E_{inital}$$

$E_{initial}$: Energy consumption by Initial model
$E_{final}$: Energy consumption by Final model
$E_{saving}$: Energy savings by the Initial model
$acc$: Accuracy of the Initial model

### 4.2.3 Confidence interval disparity

We observe a significant disparity in the average confidence level per class in the output of the Initial model. An example of this could be when training in Cifar10 dataset with a colour filter, we observed an average confidence level 17% for the cat class while at the same time we saw an average of 60% confidence when classifying ships.

This result is observed because during the colour filtering in the above case, we eliminate some colours from the picture that do not meet the filter criteria. This specific case can be explained as in the dataset we find many different shades of cats in diverse backgrounds whereas ships are usually of a similar colour in the ocean. Therefore, we can divide the classes into 2 categories; one with high average confidence interval and others with low average confidence interval.

Thus, we can argue that specific filters are better for classifying specific classes (with higher confidence interval) or group of classes. While at the same time feature classification could give wrong results with images sharing a similar attribute. This can be observed with the case of ship and aeroplanes both having blue backgrounds. If we extract the blue colour from the images, it would be trivial to understand how the Neural Network can misclassify the objects with high confidence.

Therefore we can conclude that the feature extraction model (Initial model) can act as an excellent pre-filter to classify into groups of classes (which share the same attributes) before a separate model does the final classification of them. This form the basis for our optimised design.

### 4.2.4 Optimised Design

Using the insight from the previous section, we re-design our algorithm to change the role of Initial model. We now use Initial model primarily to identify if the image belongs to a group of classes. The conditions should be that it is classified to a class with a high average confidence interval for the Neural Network and the confidence interval for the specific image should be higher than the set threshold.

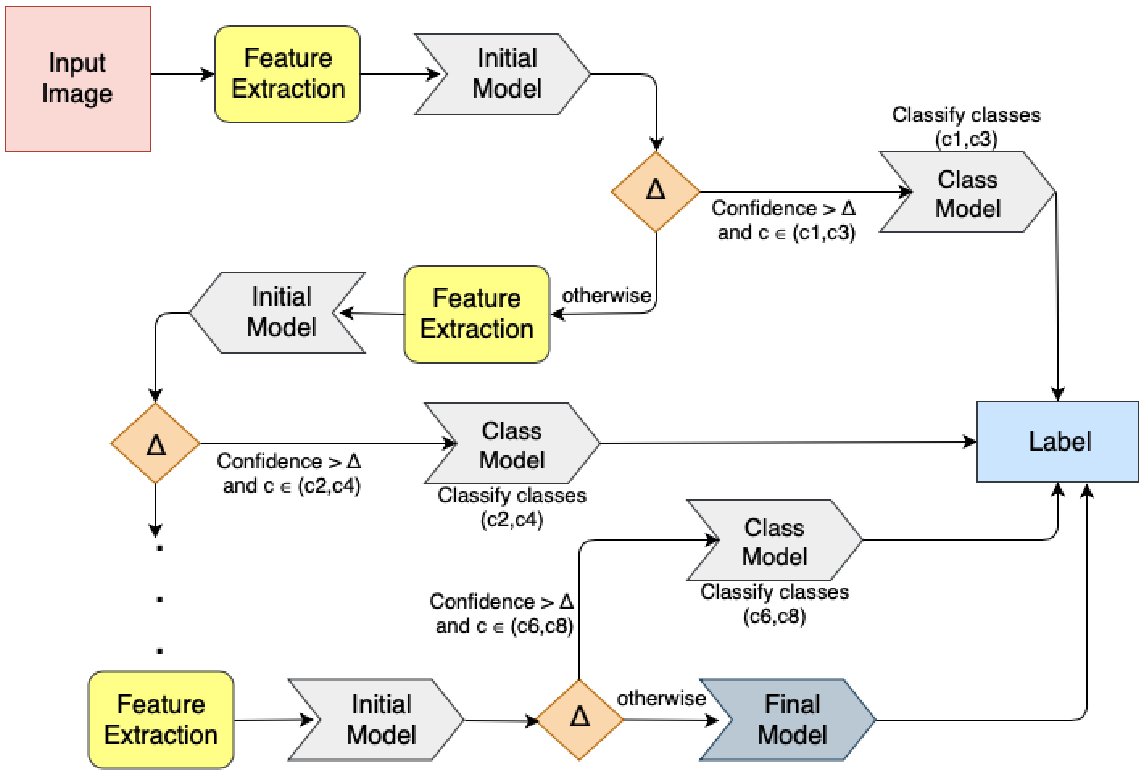


Figure 4.13: Optimised Design of the algorithm with multiple Initial models

As shown in Figure 4.13, if the image is classified to this specific group, it is classified into the exact class by the Class model. Class model is only required when a group has more than 1 class. It is to be noted that the class model would be far less complex as compared to Final model because it only needs to classify between a small set of classes.

If the image doesn't meet the criteria, it is passed on to the next feature extraction and Initial model. This is repeated until we reach the Final model for classification. The number of feature extraction layers would be decided by the energy-saving goals as described by the equations in Section 4.2.1.

Unlike the Initial design, we would choose features not only based on the accuracy of classification but the number of classes which have high average confidence interval as well. This is because our aim is not to classify directly with Initial model but to classify it as a group of possible classes. A feature with many classes with high average confidence interval indicates that this feature is shared between a lot of classes. Thus by choosing such a feature, we can extract away a broad set of input image away from the rest of the models. Note that we have assumed that the confidence interval is chosen large enough that it doesn't allow the majority of the classes to form a group, as this might defeat the purpose of the Initial model.

Finally, it could be argued that this novel approach is better than the traditional cascading models like conditional deep learning classifier [13], Cascading Neural Network [37], the Distributed Deep Neural Network [55] as it uses the semantic data to have low energy classification as compared to training a model for full input space. While the tree approaches such as [58] and [59] often try to classify the input into groups by using simple classifiers. There are a few limitations with this design as the final accuracy depends a lot on the first few classifiers which might often misclassify due to their simplicity, moreover, in case of low confidence the image is directly passed to the Final model incurring substantial, energy penalty. While in our design, we adopt a positive activation pattern at the initial level, which only classifies the model in branch if it has high confidence. If the model fails to meet the criteria, it is passed to the next feature rather than using Final model directly.

## 4.3 Results and Evaluation

In this section, we discuss and evaluate the results of extracting different semantic data from the input, as shown in Section 4.1. We also evaluate the performance of our algorithm and observe the effect of the optimisation.

### 4.3.1 Initial model with semantic data

We extracted many different types of semantic data; colour, texture, edges, corners and even performed systematic dimension reduction to reduce the size of input space. We would compare the effect of this extraction with multiple parameters. This would help us choose the appropriate semantics corresponding to the specific dataset for the algorithm discussed in Section 4.2.

#### Colour

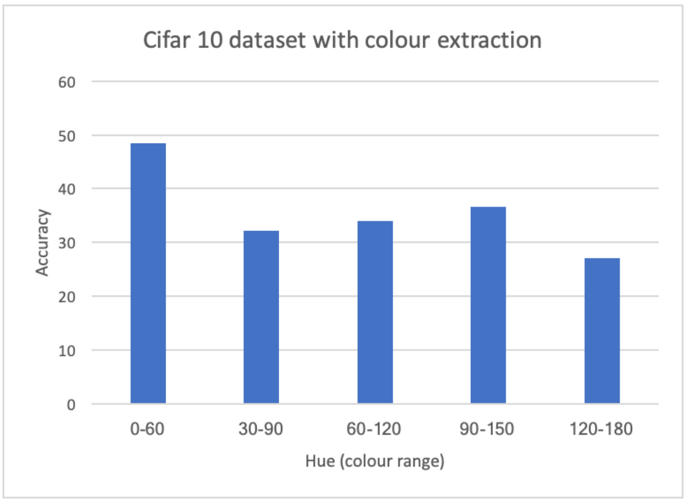


Figure 4.14: Accuracy of Initial model with colour extraction of different ranges

In Figure 4.14, we observe a significant variance in the accuracy of different colour ranges. This can be associated with the choice of classes and dataset. The highest accuracy achieved is 48.5% for 0-60 range, and the lowest is 27.1% for range 120-180. Note, we have used overlapping ranges for colours. This is acceptable as our goal is to limit the colours as the output for the image with respect to the image. As long as we can provide limited colours within a range, we are expected to save energy. This is evident in Figure 4.15, where we observe average energy consumption from 110 to 144 Joules. This includes the 9 Joules average of energy required for semantic extraction. This is still lesser than both the Initial model and Final model in established approaches discussed in Section 3.2.

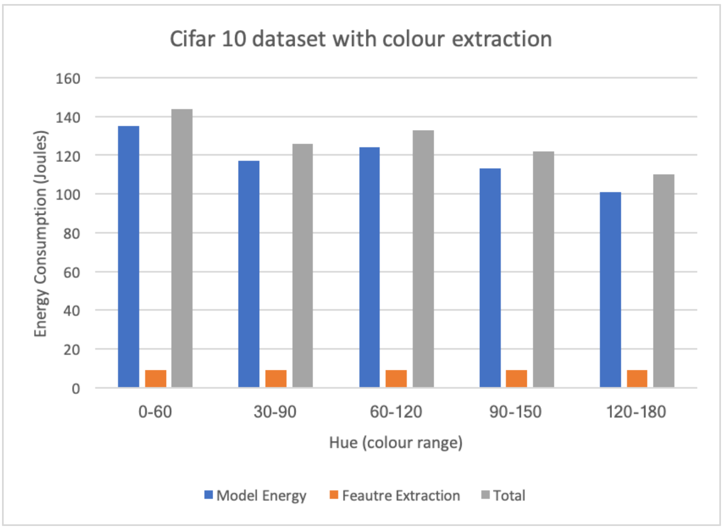


Figure 4.15: Energy Consumption of Initial model with colour extraction of different ranges

**Texture**

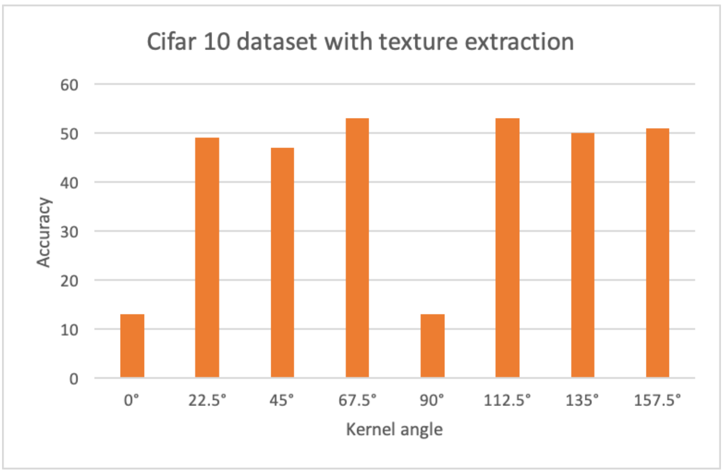


Figure 4.16: Accuracy of Initial model with texture extraction of at multiple angles

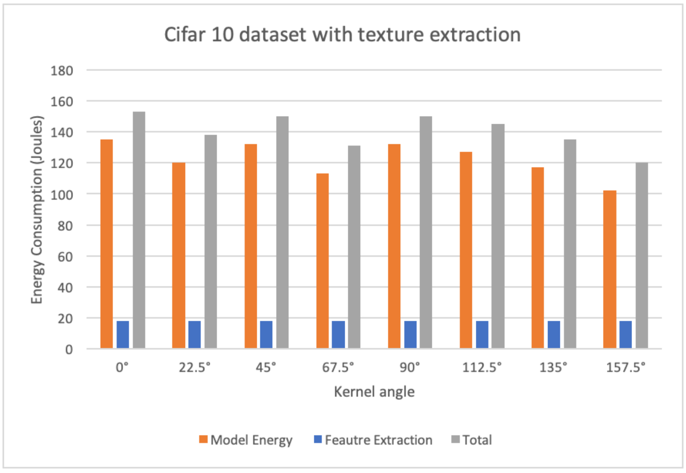


Figure 4.17: Energy Consumption of Initial model with texture extraction of at multiple angles

Figure 4.16 shows the accuracy of a simple (Initial) model trained on different textures extracted from the input image. We observe a significant difference, with 13% to 51% in the accuracy of the network. This is caused due to the set of input data and the labels attached to them. Figure 4.17 shows the energy consumption for the extraction of textures. Including the 18 Joules of energy spent in extraction, we observe total energy consumption from 120.8 Joules when the kernel is aligned at 157° to 153 Joules when the kernel is aligned at 0°.

### Edges

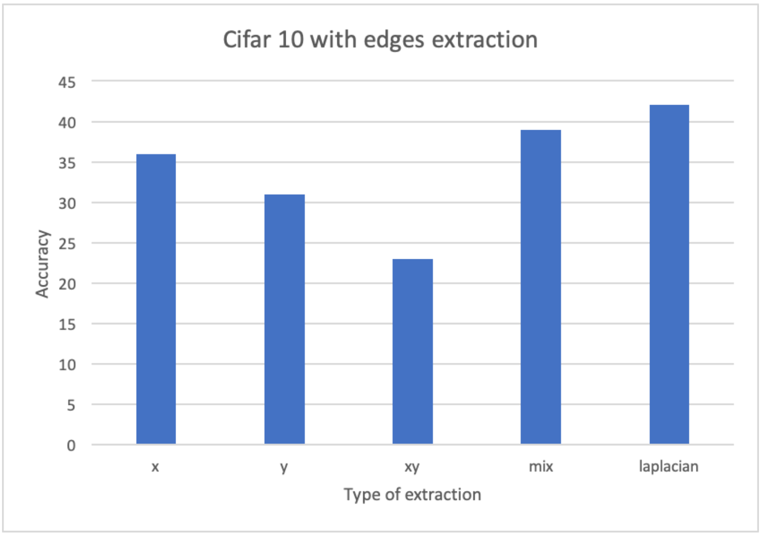


Figure 4.18: Accuracy of Initial model with different types of edge extraction

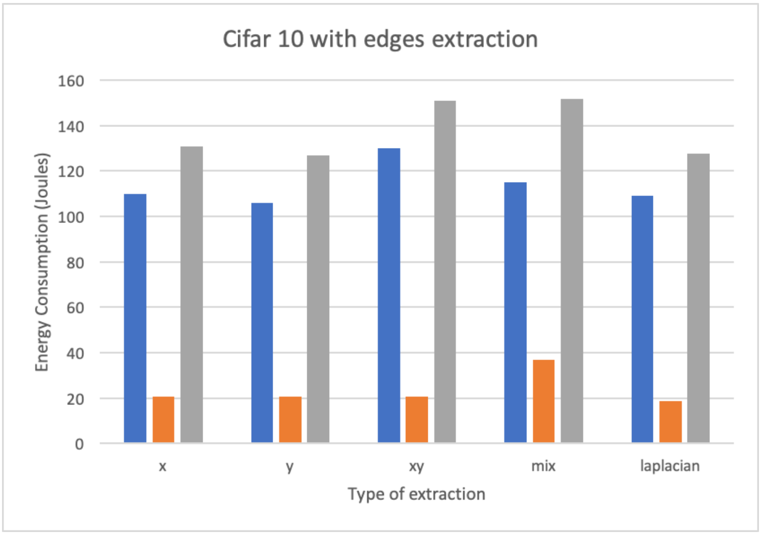


Figure 4.19: Energy Consumption of Initial model with different types of edge extraction

We have adopted a few techniques for edge extraction from the images. These include extracted only vertical direction (denoted by x), vertical and horizontal direction (denoted by xy), adding vertical and horizontal weights (denoted by mix) and using Laplacian (denoted by laplacian). In Figure 4.18, we observe accuracy from the range of 23% for xy to 42% for laplacian. Figure 4.19 shows the total energy in each type of extraction. This includes both the model energy consumption and feature extraction energy consumption.

In this case, we observe the variance in even the feature extraction energy consumption as different techniques are adopted to extract edges.

### Corners

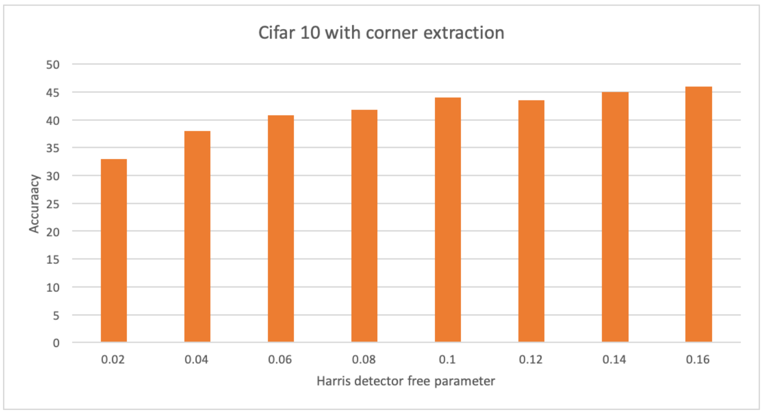


Figure 4.20: Accuracy of Initial model with different free variable values in Corner Harris extraction

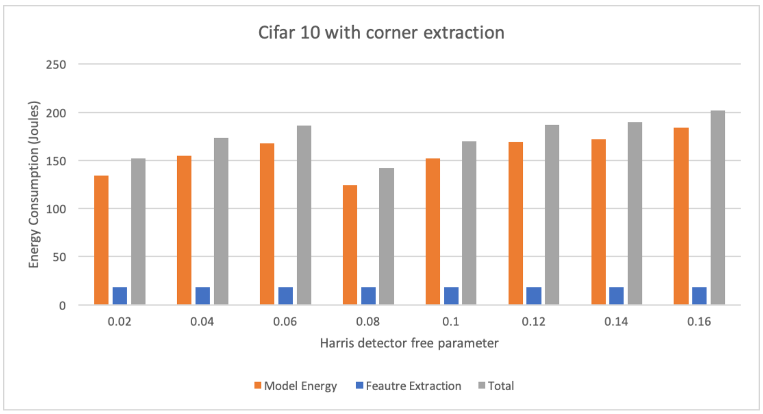


Figure 4.21: Energy Consumption of Initial model with different free variable values in Corner Harris extraction

We use the Corner Harris for corner extraction. The window size is fixed at (3*3); this is chosen with respect to the image quality and size [45]. Thus we vary the value of equation with free parameter. We observe a range of accuracy from 32% to 46 % for different values of the parameter, as shown in Figure 4.20. Further, energy consumption also varies significantly with the choice of the free variable. Figure 4.21 shows the total energy varies from 142 Joules for 0.08 value of the parameter to 202 Joules for 0.16 value.

### LDA

For Systematic dimension reduction, we use LDA or Linear discriminant analysis, explained in detail in Section 2.6.1. Figure 4.22 shows the accuracy achieved with different output dimensions. It can be surprising to observe that the accuracy doesn't improve much with increasing the output dimension as it should intuitively. The reason behind this can the that using more dimensions doesn't provide more information (any significant amount of information) for this dataset; thus, we observe no improvement in the accuracy.

Figure 4.23 describes the energy used both during converting images to lower dimension and energy consumption used by the neural network model (Initial model). The net energy usage by this technique is significantly less as compared to other semantics techniques and n both the Initial model and Final model in established approaches discussed in Section 2.5. We observe a small variance in energy usage from 35 Joules to 44 Joules for different dimensions. The general trend is that energy consumption increases with more dimensions.

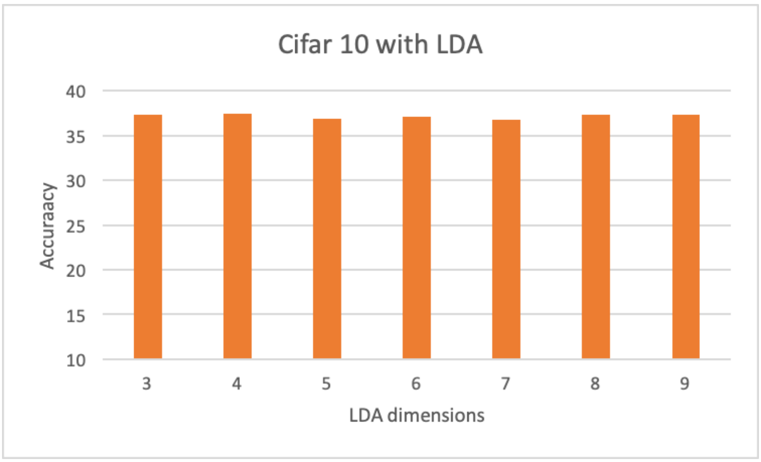


Figure 4.22: Accuracy of Initial model with different output dimensions in LDA

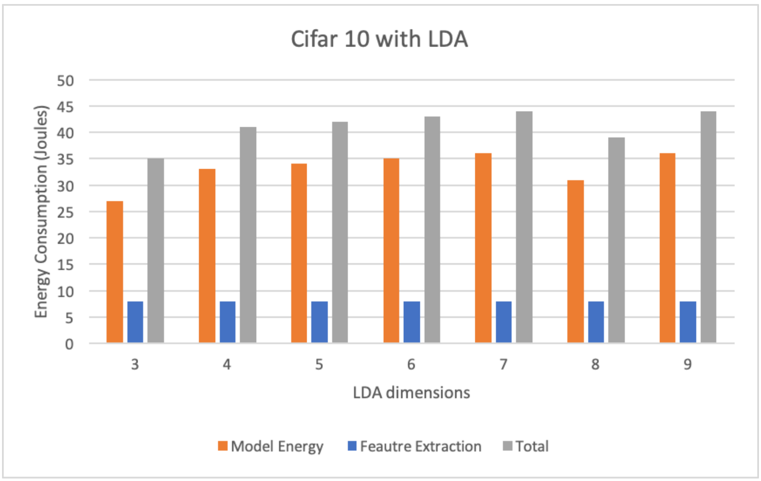


Figure 4.23: Energy Consumption of Initial model with different output dimensions in LDA

To conclude, we observed a wide variety of semantic data. For our data-set, some semantics such as texture provided high accuracy while others like edges are not very accurate on average. We also observed a substantial difference in energy consumption, eg. LDA on average only uses up-to 44 Joules of energy while corners on average are using 175 Joules of energy. We also observed large variances with different parameters in many semantics, therefore choosing the right parameter along with the appropriate semantics is required to good achieve energy savings. Consequently, we proposed algorithms to choose and arrange among these semantic models to achieve comparable accuracy to the established (in Section 3.2) models while saving energy.

### 4.3.2 Proposed Algorithm - Initial Design

We designed this Initial design by replacing the Initial model with a similar model that uses the semantic data to classify images (for more details, refer to Section 4.2). The main novelty in this algorithm would be how we choose the semantic models to reduce the overall energy consumption. In this section, we would compare this algorithm to the established algorithms discussed in Section 3.2.

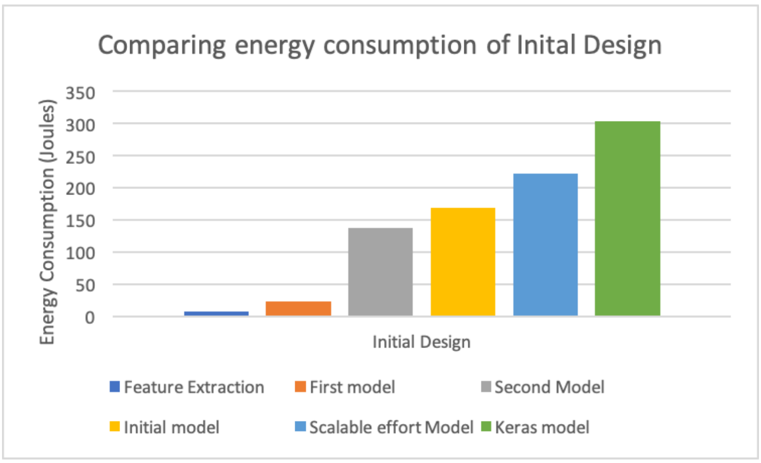


Figure 4.24: Comparing Energy Consumption of Initial design with established algorithms

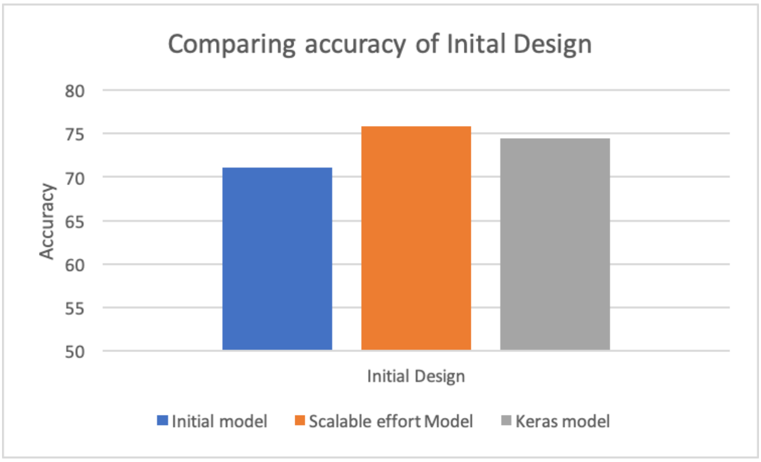


Figure 4.25: Comparing Accuracy of Initial design with established algorithms

Figure 4.24, shows the breakdown of the energy consumption by the initial model and compares it with Keras model and Scalable effort model [12] (discussed in Section 2.5). The cost of the Initial design (Total cascading model) is calculated by summation of energy usage during feature extraction and energy used by the Initial (First) and Final (Second) models. Overall, we observe an improvement of a 23% improvement over the Microsoft implementation (Scalable effort classifiers) and a massive 44% decrease in energy consumption when compared to the single model (Keras example model).

However, Figure 4.25 shows that the cascading model has reduced accuracy when compared to both the Single Model and Microsoft design (by 3% and 4.7% respectively). This is understandable as we have replaced the Initial model with a simpler model that understands the semantics of the data. To counter this effect, we have further developed the optimised version of this algorithm, which provides higher accuracy.

### 4.3.3 Proposed Algorithm - Optimised Design

In this design, we redefined the role of Initial model as a filter to detect if the input belongs to a set of class. Then the Class Model was responsible or classifying the input to its exact class. This model reduces our reliance on Initial model, which is often simple in design and can lead to incorrect classifications.

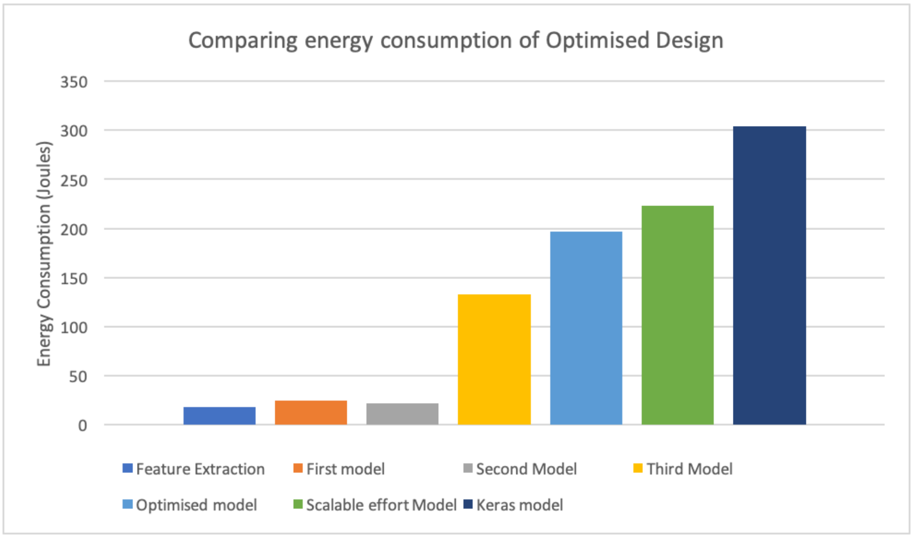


Figure 4.26: Comparing Energy Consumption of Optimised design with established algorithms

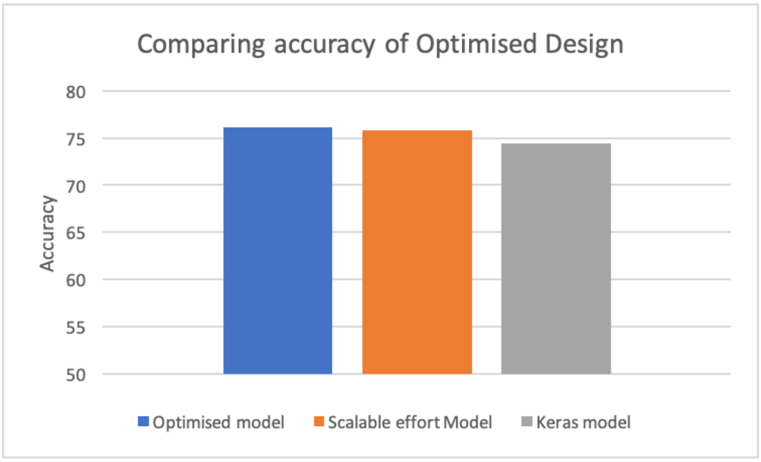


Figure 4.27: Comparing Accuracy of Optimised design with established algorithms

Figure 4.27 compares the accuracy of the Optimised design with the established models. We observe that the Optimised model performs slightly better by 0.5% and 1.5% with respect to Scalable effort classifier and Keras model. Thus, the Optimised design was able to address the limitation of the Initial design.

When we compare the energy consumption in Figure 4.26, we observe that energy consumption of Optimised design (which is the summation of First, Second and Third cascaded models) is 13% improvement over the Scalable effort classifiers and substantial 35% improvement over the Keras model. One important observation can be that in both Initial design and Final design, the energy consumption by the Final model (Second and Third model respectively) was substantial (80% and 67%) as compared to the low-cost

Initial models.

If we compare these numbers to the Initial model, we observe that the energy consumption of the Optimised model has increased primarily due to the introduction of a new model. While, on the other hand, we find the accuracy to be improved as well. Thus this imposes the classic energy vs accuracy trade-off. If we need to match the accuracy of the established models, then we need to be satisfied with lesser energy savings. This decision as to which models to choose would depend on the exact use-case.

To conclude, in this chapter, we extracted semantic data from the input via multiple techniques. Then we created algorithms to choose and arrange the models which use this semantic data to classify the input. On the one hand, we observed the energy improvements such as 35% on the Keras model maintaining similar accuracy, while on the other hand, we also saw significant variance in the results of semantic data output which might pose a limitation to this technique as choosing a semantic becomes less reliable.

Another observation was the disproportionate consumption of energy by the Final model as compared to low-cost Initial models. We will address this challenge in the next chapter where we try and limit the energy consumption by these Final models.

# Chapter 5

# Energy constraint Bayesian Optimisation

Until now, we have discussed techniques to group classes and create the models for partial classification in the cascading chain. We observed that if we arrange these partial classification models by the given algorithm in Section 4.2, we can save net energy usage. In this chapter, we would discuss in detail the methods used to improve the 'Full model' or the 'Final model' in the cascading chain used for full classification. This model is designed as the last resort if the 'lower-dimensional' models (also known as Initial models) are unable to classify. Therefore we typically expect a high accuracy and energy usage from this model; however, this model still contributes significantly towards energy consumption as seen in the last Section 4.3. This chapter will look to maximise the efficiency of the model given an energy constraint.

In this chapter, we would begin by discussing the initial idea to optimise the model in Section 5.1, followed by the design and implementation details of the algorithm in Section 5.2. Further, we would detail the additional optimisations in Section 5.3. Finally, we would discuss and evaluate the results in Section 5.4.

## 5.1 Initial Idea

All the traditional approaches discussed in Section 2.5.3, used the state-of-the-art established model as the final classifiers in cases of the failure by partial classification models. These classifiers are generally designed by experts with some intuition; however, most of them did not follow a systematic process while choosing the 'magic values' in these models. Moreover, often, these established models did not account for energy utilisation and primarily focused on the accuracy or reducing the error rate.

Rather than choosing an established model, we developed an algorithm to systematically choose the 'Final model' hyper-parameters to give the best accuracy, while accounting for energy usage. We used a popular technique called 'Bayesian optimisation' to tune our hyper-parameters to optimise for accuracy while providing an energy-constraint (explained in Section 2.7). This technique allowed us to effectively search the input space for all the possible models and choose the most accurate one within the given constraint. This technique was proved useful in a similar context in a recent study named HyperPower [75].

However, we realised that these hyper-parameters were still limited. It assumed many parameters, such as a 'number of layers' and 'kernel size' (Kernel size in CNN represent the height and width of the 2D convolutional window). To counter this limitation, we went further and developed a dynamic model with little or no assumption about the structure of the given model before optimisation. This new algorithm resulted in an assumption of fundamental parameters such as activation function, number of iterations and type of hidden layers which can be safely assumed given the kind of data and computational resources. Thus we performed Bayesian optimisation with the following parameters; the number of features, kernel size, number of layers, learning rate, weight decay.

Another essential idea was to decouple the acquisition function from external constraints, i.e. energy constraint. This allowed us to take real-world run-time energy readings rather than predicting the energy consumption from the parameters of the Neural Network. This is important because as we discussed in Section 2.5.1, energy consumption by the Neural Network is largely effected by RAM and data-movement. This restriction can make prediction made by the structure of Neural Networks inaccurate.

## 5.2 Algorithm Design

The architecture of this system revolves around the process of Bayesian Optimisation, as described in Section 2.7. Initially, we get the required data followed by modelling the posterior probability distribution with the given constraints. This is followed by applying the acquisition function to get the next point. This point is evaluated on the objective function to get the required value. The above process is repeated until we get run out of given iteration budget.

Figure 5.1 represents the detail design of the algorithm. The algorithm has the following main steps:

1. It starts by the user defining the bounds of each hyper-parameters, 'iteration budget' and the 'energy-budget' for the optimiser. These parameters help determine the kernel space that the optimiser can explore to find the optimal model. Users have to careful while choosing these limits as choosing conservatively or choosing with preexisting biases can result in sub-optimal models while choosing too generously might result in long run time to get efficient model.

2. Next, the optimiser randomly chooses these set of values of these parameters from the defined limit. These values are used to take a random reading of the model using the objective function. The goal of this step is to explore the entire input space to obtain a good start point for optimisation. It helps diversify the exploration space and improve the speed to find the optimal model, as shown in the experiment [76].

3. After getting these initial reading, we create the posterior distribution of the expected objective function with the gained information. This distribution is generated using the regression of Gaussian processes with the assumed kernel and Gaussian function, as explained in Section 2.7.2.

4. This generated model is passed to the acquisition function, which provides the next point to explore in the given distribution.

5. This point (representing the new parameters in the input space) is passed to the **Evaluation process** as described in Figure 5.1. The evaluation is a three-step process, where firstly we construct a new Neural Network based on the 'iteration parameters' which defined the point output by the acquisition function. Secondly, we train this Neural Network with the training data to the number of epochs defined. Lastly, we measure the accuracy and energy of this model on the test dataset.

6. Therefore we get the exact value of this new point. This process of creation of posterior distribution, finding next point and evaluating this point to gain more information about the model is done in the loop by the optimiser until we the consume the entire 'iteration budget', which defines the number of iteration allowed by the optimiser as shown in Figure 5.1.

7. Finally, we get the best result from the optimiser as output.

8. These parameters are then used to train the final output model. At this stage, we can afford to train this output or 'Final model' with more epochs to get the accurate optimised results.

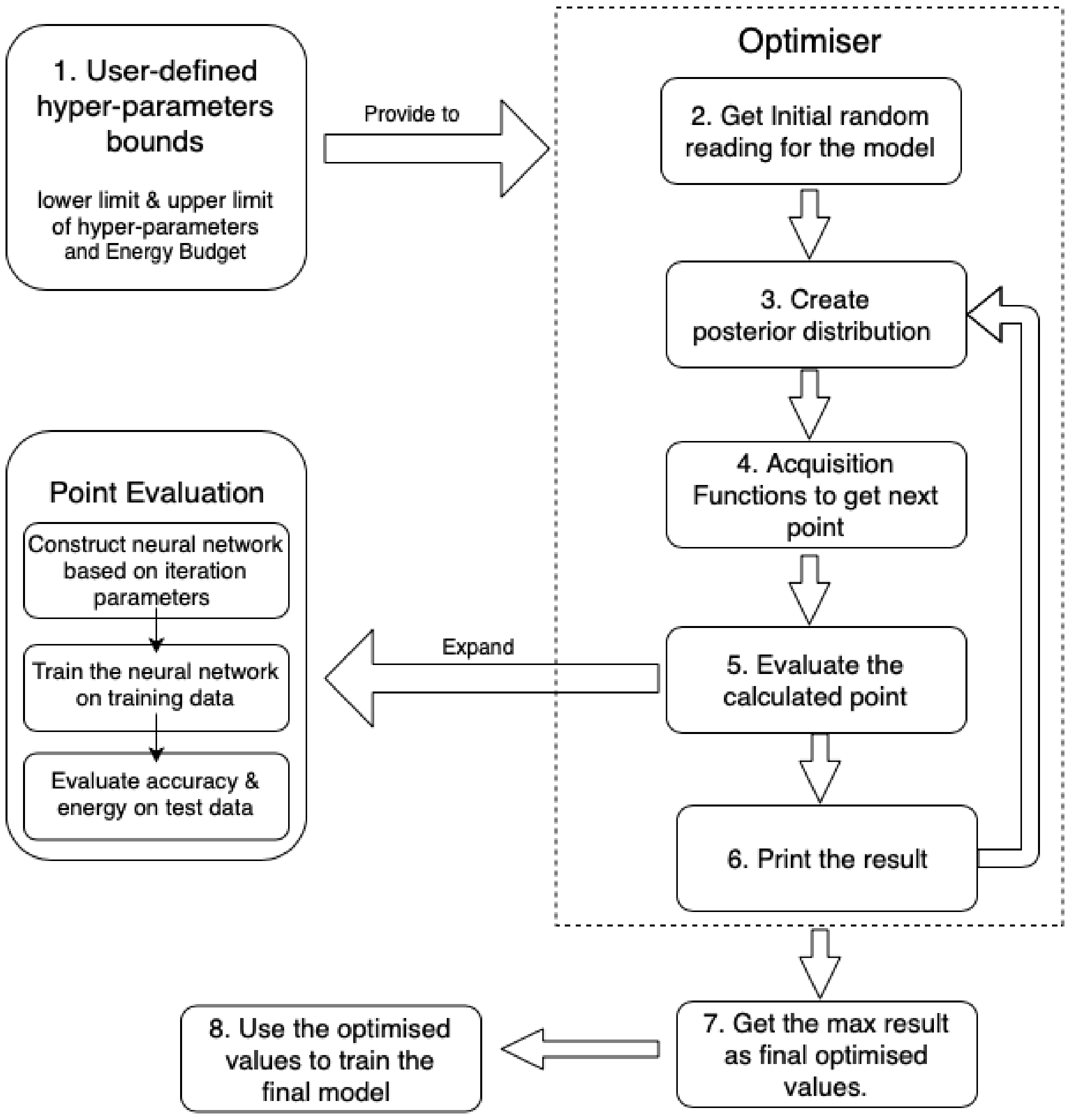

1. User-defined hyper-parameters bounds
lower limit & upper limit of hyper-parameters and Energy Budget
Provide to
Optimiser
2. Get Initial random reading for the model
3. Create posterior distribution
4. Acquisition Functions to get next point
5. Evaluate the calculated point
6. Print the result
Expand
Point Evaluation
Construct neural network based on iteration parameters
Train the neural network on training data
Evaluate accuracy & energy on test data
7. Get the max result as final optimised values.
8. Use the optimised values to train the final model

Figure 5.1: An overview of the Algorithm Design

Now, we will discuss the main challenges and decisions with respect to the above design in more detail.

### 5.2.1 Choosing Acquisition Function

As we discussed in Section 2.7.3, the acquisition function help to choose the next point to explore in the posterior distribution. When choosing where to evaluate next, we have two opposing goals. One one hand we would like to incentives exploration of neighbourhoods which have not been explored to get most optimal value from the explored regions, while on the other hand, incentives exploitation of areas we have high confidence will have an optimal value. More rigorously, we can think of exploitation as seeking places with high mean, and exploration as seeking areas with high variance.

The choice of acquisition function is one of the critical determinants of success when using Bayesian Optimisation. The decision should ideally depend on the nature of hyper-parameters and optimisation itself. Therefore, often, the default acquisition function is not the best choice.

Out of the three most popular Acquisition Function discussed in Section 2.7.3, the expected improvement is proved to be the most popular and best performing [62]. Expected Improvement (EI) is better-behaved than probability of improvement (PI), but unlike the method of upper confidence bounds (UCB), it does not require its own tuning parameter. In recent study 'Practical Bayesian Optimization of machine learning algorithms' [62], the EI has proved to be the optimal in fewer than half as many evaluations vastly outperforming both the PI and UCB.

Despite these results, we have performed our experiments to compare the acquisition functions in Section 5.4.2. We found that with given our datasets the Expected Improvement (EI) outperforms the other acquisition functions. This is keeping the kappa ($\kappa$) constant, thus keeping the same balance in 'Exploitation vs Exploration'.

### 5.2.2 Defining Objective Function

Next, we will define the objective function or the 'black box function' to evaluate the exact value depending on the given parameters. In our case, this black box would be a Neural Network which would need to be optimised given the constraints. Thus for every given configuration, we would have to train a Neural Network on the training data and then evaluate it to get the accuracy on the test data set. Mathematically we can express our function in the following as:

$$ObjectiveFunction \equiv accuracy(f, D)$$

subject to,

$$\mathcal{E}(f(D)) \ < \ e$$

$$l < h < u \qquad \forall h \in H$$

where,

$$f \equiv Network(D', H)$$

$f$: Trained Neural Network
$D$: is Test DataSet
$D'$: is Training DataSet
$H$: is User Defined hyper-parameters
$l$: is lower-limit of hyper-parameter
$u$: is upper-limit of hyper-parameter
$e$: is max energy budget
$\mathcal{E}$: Energy measurement function

The parameters are the hyper-parameters explored by the Bayesian algorithm. They are *conv1_features, conv1_kernel, conv2_features, conv2_kernel, conv3_features,* *conv3_kernel, full1_features, decay_rate, lr.* We are currently assuming 3 convectional layers, 1 dynamic fully connected layer and an output layer with the fixed size of the number of output classes. We would vary a number of layers in the further sections. Further explanation of each parameter is explained in the table below:

| **Hyper-Parameters** | **Explanation** |
|---|---|
| conv1_features | Dimensionality of output space of convolution layer 1 |
| conv1_kernel | Height & Width of 2D convolution window of layer 1 |
| conv2_features | Dimensionality of output space of convolution layer 2 |
| conv2_kernel | Height & Width of 2D convolution window of layer 2 |
| conv3_features | Dimensionality of output space of convolution layer 3 |
| conv3_kernel | Height & Width of 2D convolution window of layer 3 |
| full1_features | Dimensionality of output space of fully connected layer |
| decay_rate | Float represent the Learning rate decay over each update. |
| lr | Float used in the optimiser to represent Learning rate. |

Table 5.1: Table explaining the hyper-parameters used in the Neural Network

The lower-limit and upper-limit are conservatively defined by the user to set the exploration range. Further energy-level is also provided by the user to limit the number of possible solutions and find the best architecture in the given energy level. Other constant parameters which do not change between experiments include batch_size (usually set to 64), and epochs (typically set to 30) are set based on convenience and similar type of input data models [?].

Once we have the trained network based on the above parameters, we use this network to evaluate the accuracy of the network (correct classifications). We also take the energy readings during this evaluation phase rather than measuring the training energy. The rationale behind this decision is that the training is potentially done once per network; therefore, we can afford to spend a large amount of energy. However, the evaluation and usage of the network is performed potentially many times and therefore needs to be optimised to get real-world impact.

Finally, all the above information (including the energy measurement) is abstracted away in the objective function, and only the accuracy is returned as the final output for optimisation. As we discussed before, in Section 5.1, this allows us to take real-world run-time energy readings independent of the external optimisation to get more accurate results.

### 5.2.3 Adding Energy Constraint

After developing the basic system to optimise the hyper-parameters, we need to add the energy constraint to restrict from choosing from the energy-intensive model. This is applied as a hard stop when the model exceeds the ‘energy-budget’. To abstract away the energy consumption from the acquisition function, we have in-turn incorporated the energy restriction in the objective function itself. This can be expressed in the equations below:

$$ObjectiveFunction = \begin{cases} accuracy(f, D), & \text{if } \mathcal{E}(f(D')) \leq e \\ 0, & \mathcal{E}(f(D')) > e \end{cases}$$

$f$: Trained Neural Network
$D$: is Test DataSet
$D'$: is Training DataSet
$e$: is max energy budget
$\mathcal{E}$: Energy measurement function

If we assume a group of models, then the objective function would return the true accuracy based on the test set until the energy-constraint is reached. However, if any model breaks the constraint, the objective function would return the minimum value of accuracy (which is 0 in this case). The basic idea being that we want the Bayesian model to explore the possible models within the constraint and not encourage any exploration beyond the 'energy-budget'. This can be better understood by the following graph.

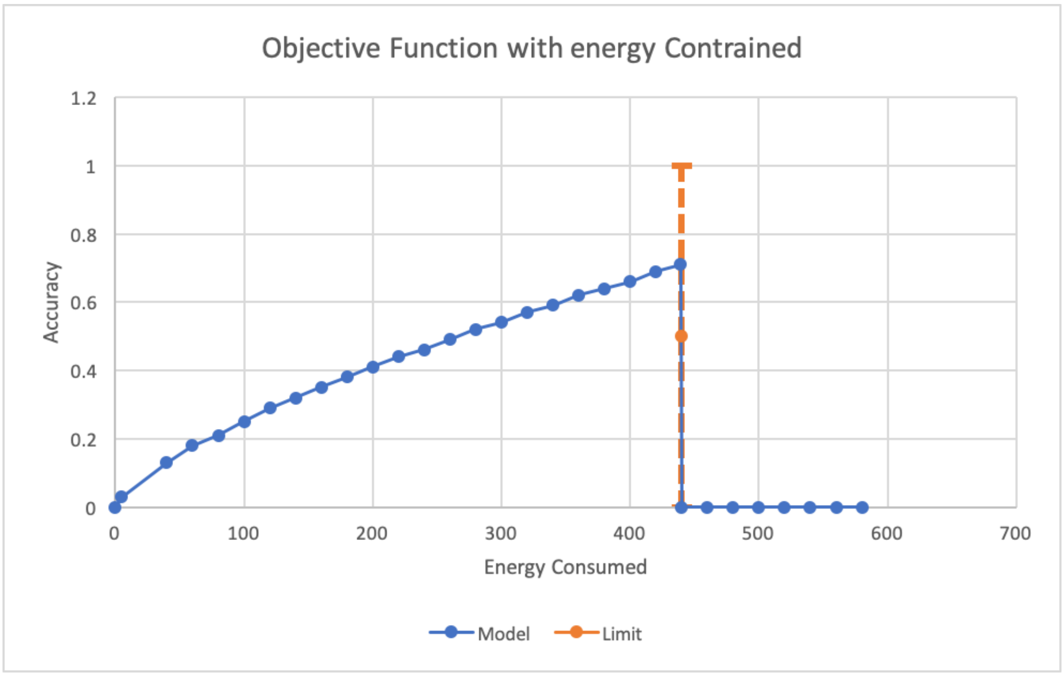


Figure 5.2: The graphical model illustrating the energy constraint with respective to the objective function

In Figure 5.2, the 'budget-constraint' is set to 440 Joules. We can observe a general trend as the model consume more energy it can afford to be more complex and thus offer better accuracy. However, as we reach the budget, the accuracy drops to 0.

### 5.2.4 Converting to discrete parameters

Bayesian optimisation with Gaussian processes has an underlined assumption that the objective function is a continuous function. However, were bound to be in a situation where some of your function's parameters may only take on discrete values. In our cases, we have many parameters (such as 'conv1_features'), which only take discrete values.

Unfortunately, due to assumptions of the Bayesian Optimisation as discussed in Section 2.7, there is no easy/intuitive way of dealing with discrete parameters. Therefore we use one of the most common techniques [77], which is to round the suggested variable value to the closest integer before evaluating the objective.

```
conv1_features = 2*int(round(conv1_features))
conv1_kernel = int(round(conv1_kernel))
conv2_features = 2*int(round(conv2_features))
conv2_kernel = int(round(conv2_kernel))
conv3_features = 2*int(round(conv3_features))
conv3_kernel = int(round(conv3_kernel))
conv4_features = 2*int(round(conv4_features))
full1_features = 2*int(round(ip1_output))
```

Listing 5.1: Code snippet converting continuous to discrete parameters

In Listing 5.1, the package implementation of the system is described. We first round the continuous parameter to the nearest integer value and cast it to the integer. Note that we have multiplied the convolutions features parameters by 2 to increase the sensitivity towards the changes in the parameter, which result in quicker exploration of the entire input space. This will give the results with lower resolution and does not changes the fundamental effects.

### 5.2.5 Dynamic Model Creation

Now, we have all the required components for our system. We have defined our Gaussian model, Acquisition Function and objective function with energy constraint. This already provides some initial results, as explained in Section 5.4.1. However, until now, we have kept the number of layers fixed. We know from the historical results [78] that deeper networks (with more layer) perform better on average than shallow networks (with less number of layers).

Keeping the number of layers constant only allows varying the number of features per for to choose a more complex model. This constraint can result in shallow models. We cannot merely increase the number of layers in our base model as it might quickly exceed the required energy consumption constraint.

The solution we came up was to have the number of layers as another hyper-parameter ('num_layers') of the model. This hyper-parameter allows the optimisation algorithm to choose vary number of layers and thus choose shallow or deep models based on energy consumption. The necessary code for the construction of the model based on the new parameter is shown in Listing 5.2.

```
f (num_layers >= 2):
    model.add(Conv2D(conv2_features, (conv2_kernel, conv2_kernel)))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

if (num_layers >= 3):
    model.add(Conv2D(conv3_features, (conv3_kernel, conv3_kernel), padding='
   same'))
    model.add(Activation('relu'))

if (num_layers >= 4):
    model.add(Conv2D(conv4_features, (conv4_kernel, conv4_kernel)))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))
```

```

if (num_layers >= 5):
    model.add(Conv2D(conv5_features, (conv5_kernel, conv5_kernel), padding='same'))
    model.add(Activation('relu'))
    model.add(Conv2D(conv6_features, (3, 3), padding='same'))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))
```

Listing 5.2: Choosing the model shape depending on the num_layers parameter

As shown in Listing 5.2, if the num_layers is chosen to be 3, we would only add one convolutional layer. Although we do need to manage the regularisation layers such as 'MaxPooling' and maintain dropout between the layers to prevent the model from over-fitting.

Another side effect of the new system was that certain variables related to higher convolutional layers (eg 'conv4_features' and 'conv4_features') would not be used in some instances such as in the case above. Their effect would only be visible based on the value of 'num_layers' parameter. However, we could argue this effect to be acceptable and as a property of the objective function, because while defining the objective function we considered it to be a 'black-box' with no constraint on its behaviour.

## 5.3 Further Optimisation

In this section, we would discuss further improvements upon the system to get better results. We would focus on the underlying intuition behind these optimisations the increment results obtained would be discussed in Section 5.4.

### 5.3.1 Elastic boundary

One of the assumption while using the Bayesian Optimisation is that the objective function is continuous, which in turn means it should have similar output values with same input dimensional values. However, we break this condition when we put artificial 'energy constraint'.

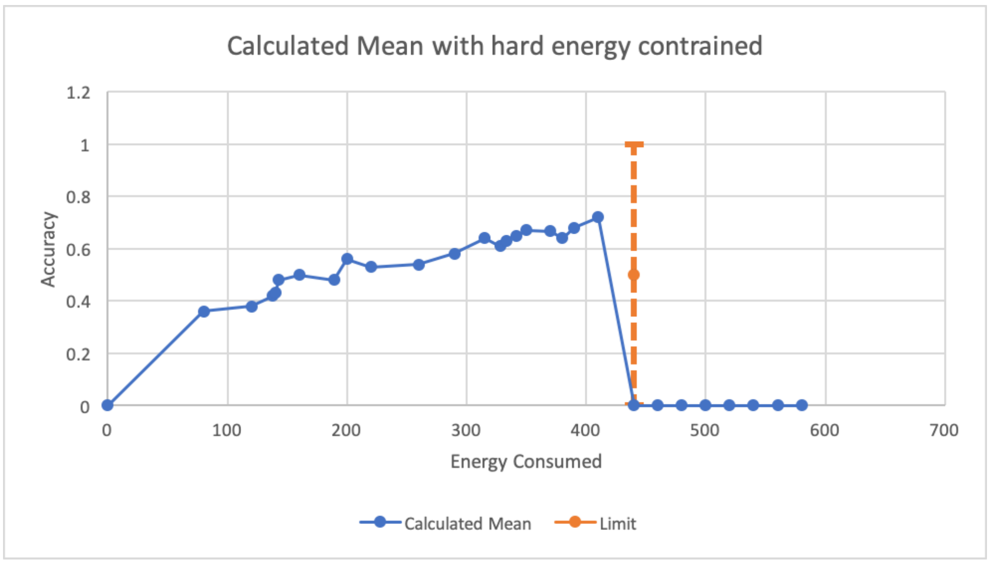


Figure 5.3: Show objective function and predicted mean with hard boundary

The effect of this phenomena is illustrated in Figure 5.3. When the optimisation algorithm explores the models beyond the energy limit, the result of the objective function is given as 0 (because of artificial energy constraint). This results in wrong mean assumption near the Energy limit as shown by the blue line. This region (near the energy limit) potentially provide the best result as it uses the entire 'energy budget' available. Further, due to the greedy strategy of the acquisition function, this region remains, often leading to sub-optimal results.

To counter this effect, we applied the technique to incrementally reduce the objective function value beyond the boundary (as shown in Figure 5.4). This results in a more continuous function with values decreasing in quadratic fashion when beyond the budget. This ensures the model receive reduced value beyond the budget but the values close to the boundary still similar to true values.

If we compare Figures 5.3 and 5.4, the result is evident that we explore the model with higher accuracy in the elastic boundary case (near 420 Joules) which lead to better models. It is to be noted that the boundary is chosen as an illustration for this case, if we increase the 'energy budget' itself, we would get better models. With an elastic boundary, we are attempting to use the entire energy budget provided to the algorithm effectively.

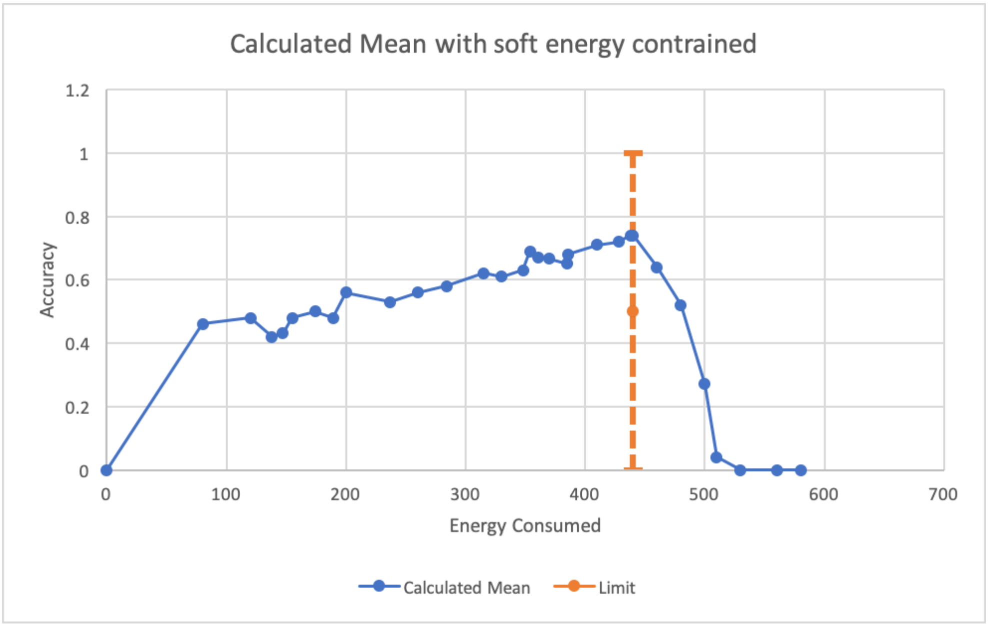


Figure 5.4: Show objective function and predicted mean with elastic boundary

We have chosen the quadratic function to reduce the objective function value beyond the energy budget due to certain advantages. Firstly, it's a continuous function and decreases rapidly as we go over the budget towards minimum objective value. While having a higher power function might reduce too quickly, defeating the purpose of the elastic boundary.

One of the drawbacks of using this technique is that in the small number of cases, we might get optimal models which are slightly beyond the 'energy limit'. We would consider it as acceptable as we do not observe significant deviation from the limit and the 'energy limit' is defined as the guideline to optimise energy and limit number of models rather than having a strict limit.

### 5.3.2 Batch Normalisation

Before training the network, we do some prepossessing to the input data (in our case, we normalise the data to resemble a normal distribution, i.e. zero mean and a unitary variance). This is primarily done to speed to improve the training speed and to prevent the early saturation of non-linear activation functions.

The same problem can be applied to hidden layers; because the distribution of the activations is continuously changing during training, a phenomenon called internal co-variate shift. This results in a slows down the training process because each layer must learn to adapt themselves to a new distribution in every training step, thus providing sub-optimal results. This can be tackled in a similar way as pre-processing by a technique called Batch Normalisation.

The input and output are shifted in the following way:

$$\hat{x_i} = \frac{x_i - \mu}{\sigma}$$

$$y_i = \gamma * \hat{x_i} + \beta$$

where,

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu)^2$$

In the above equations, $x_i$ and $y_i$ are the input and output of the layer, respectively; m is the size of the batch, $\gamma$ and $\beta$ are variables learned during back-propagation of the training stage. We observe the improvement in the overall accuracy with the introduction of batch normalisation in Section 5.4.4.

```
keras.layers.BatchNormalization (axis=-1, momentum=0.99, center=True, scale=
    True, beta_initializer='zeros', gamma_initializer='ones')
```

Listing 5.3: Batch normalisation

Keras library provides an easy interface to perform batch normalisation as shown in Listing 5.3. We have discussed the parameters with the value we are using for our implementation in Table 5.2.

| Parameters | Value Used | Explanation |
|---|---|---|
| axis | -1 (cloumns) | The axis that should be normalised (calculate mean and s.d.). |
| momentum | 0.99 | Momentum for the moving mean and the moving variance. |
| center | True | Check if we add offset of beta to normalised tensor. |
| scale | True | Check if multiply by gamma. |
| beta_initialiser | Zeros | Initial weight of beta. |
| gamma_initialiser | Zeros | Initial weight of gamma. |

Table 5.2: Table explaining the parameters used in Normalisation

## 5.4 Results and Evaluation

In this section, we discuss and evaluate the results of the major experiments mentioned. We have evaluated the results with a few different datasets to solidify our findings. Furthermore, we would also look at the various optimisations performed and their effects on the overall results.

### 5.4.1 Initial idea

In this model, we conducted an experiment with no optimisations. This experiment included a fixed number of layers, with default acquisition function and basic objective function. We would compare this result with the established model discussed in Section 3.2 to observe the full effect of our technique.

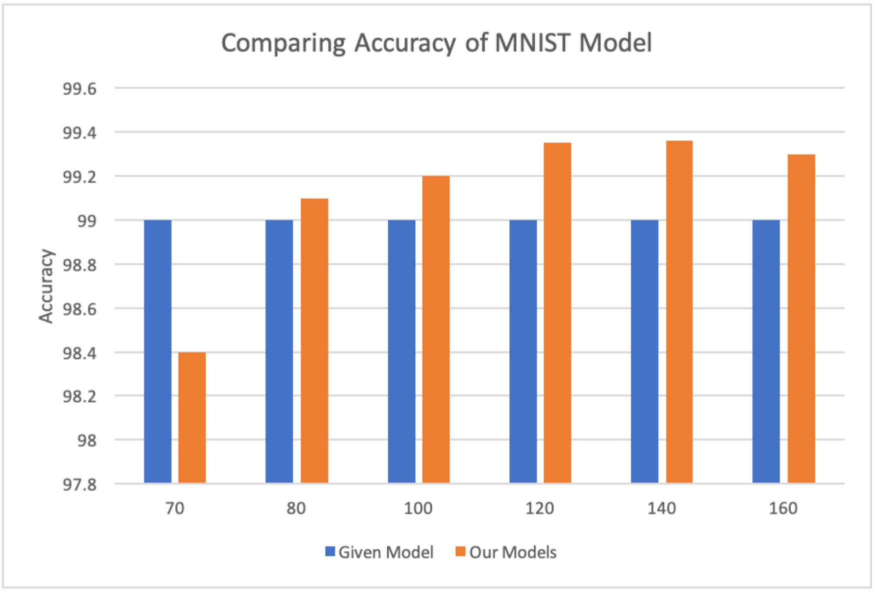


Figure 5.5: Comparing accuracy of MNIST Database with different energy budget

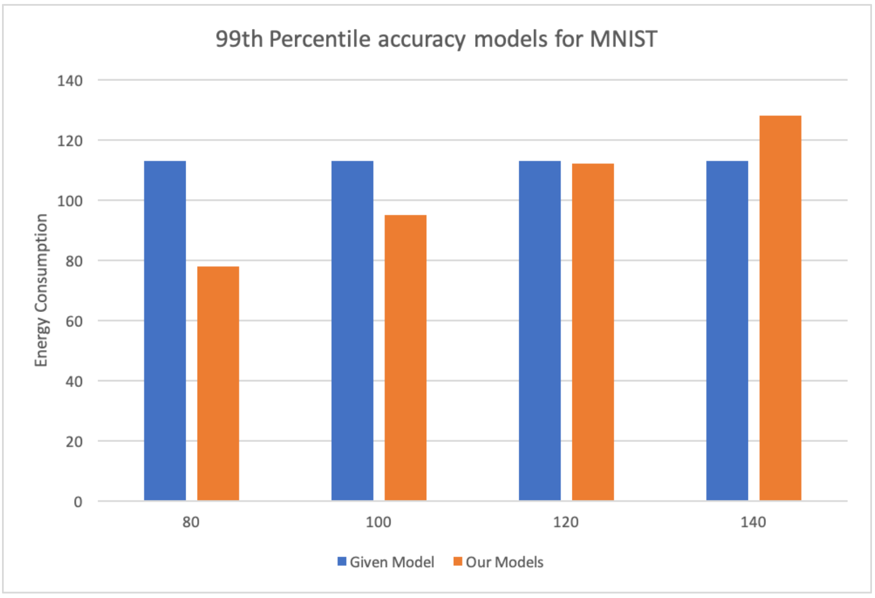


Figure 5.6: Comparing energy usage of MNIST Database with different energy budget and 99th percentage accuracy limit

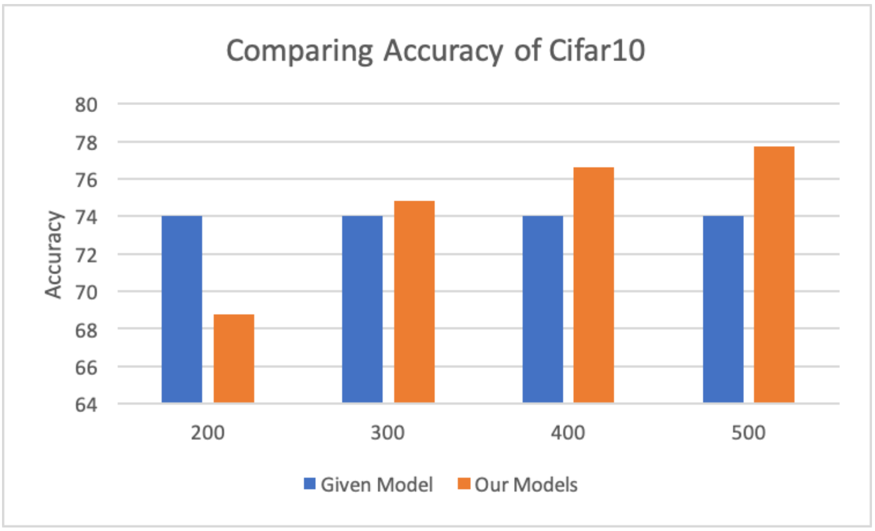


Figure 5.7: Comparing accuracy of Cifar10 Database with different energy budget

Comparing the results in Figure 5.5, we can observe no significant decrease in the output accuracy for energy budget 80 to 160 Joules. However, we find a drop in the accuracy of the network as we reduce the budget to 70 Joules. We would, therefore, discard this model as it doesn't meet the accuracy standard of the Given Model. Contrarily, choosing a model such as with 160 Joules would result in too much energy usage with little gain in the accuracy; therefore we would also ignore this model.

In Figure 5.6, we filter out models which match or exceed the accuracy of the given model. We observe that if we compare our best model with the energy of 80 Joules with the energy of the given model (113 Joules), we can observe an energy reduction of 29% as to get same accuracy (99% and 99.1%). Therefore using these results, we can safely validate the effectiveness of Bayesian optimisation.

In the case of Cifar10, we observe only a slight increase (0.4%) in the accuracy with the 300 Joules budget as shown in Figure 5.7, which is close to the 304 Joules consumption of the given model. Thus we do not gain much at this stage of the algorithm.

### 5.4.2 Comparing Acquisition Function

In Figure 5.8, we compare the three most popular acquisition functions discussed in Section 2.7.3 (Upper Confidence Bound, Expected Improvement, Probability of Improvement) while finding an optimal solution with Cifar 10 dataset. We also vary the 'Exploitation vs Exploration' parameter in each to check if they get any better performance.

Although, we observe similar end values for all acquisition functions, however still the Expected Improvement (EI) outperforms the rest as the epochs increase. Moreover, we also observe that relatively EI with $\xi = e^{-1}$ even have the quickest learning rate and reach optimal values faster as compared to other functions.

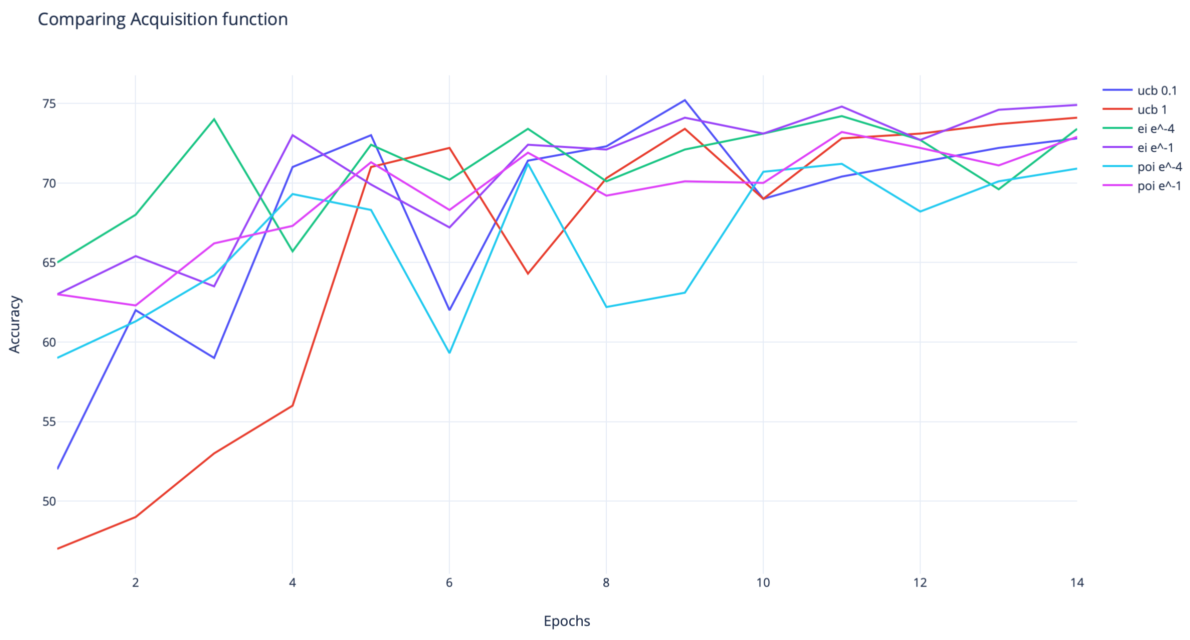


Figure 5.8: Comparing Acquisition functions

### 5.4.3 Affects of Elastic boundary

To measure the effectiveness of the elastic boundary, we can compare the results obtained by the trained models with rigid and elastic boundaries. Figure 5.9 compares the best accuracy obtained with three different energy budgets. Using Cifar 10 dataset, we find a slight improvement in the accuracy in all cases using this optimisation. We observe a 3% improvement with 200 Joules of budget, 1.2% improvement in 300 Joules budget and about 1.2% improvement in 400 joules budget.

On the contrary, we also observe no significant improvement or decline in the accuracy of the network when applied in MNIST dataset. This is primarily because of the already high accuracy rate in with the current models.

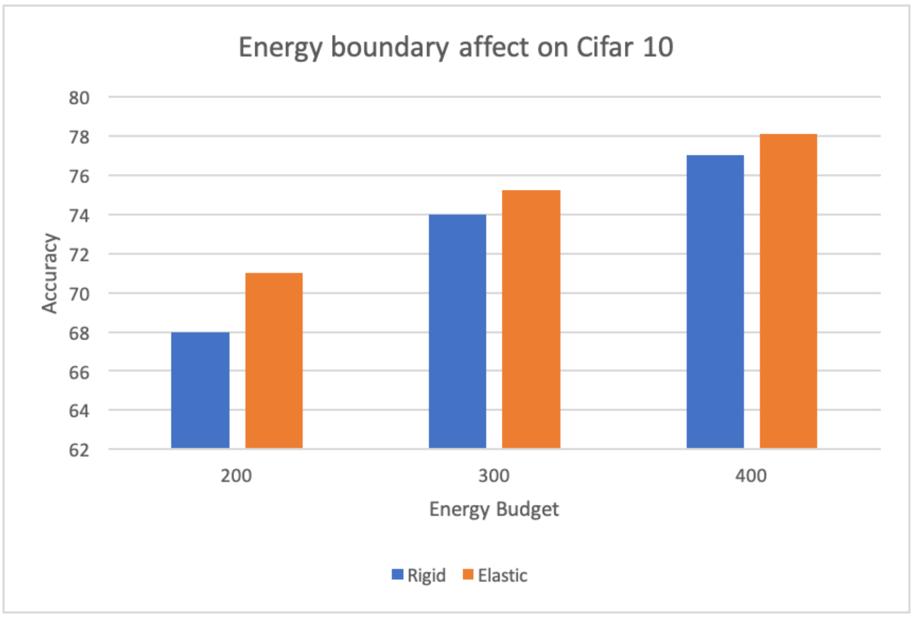


Figure 5.9: Comparing accuracy of Cifar 10 Database with elastic boundary affect

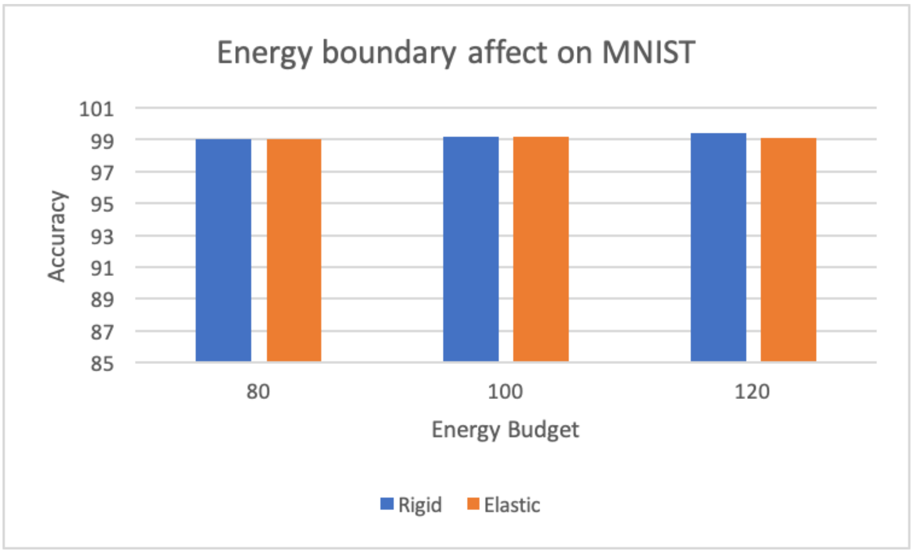


Figure 5.10: Comparing accuracy of MNIST Database with elastic boundary affect

Therefore we can conclude that elastic boundary can provide better models and thus accuracy improvement if implemented correctly. This is keeping in mind that the energy budget is not strict and just the guidelines to save energy.

### 5.4.4 Affects of Batch Normalisation

Batch normalisation normalises the input for each layer to prevent any o-variate shift. We have implemented Batch Normalisation after each convolutional layer to normalise input for each layer corresponding to layer after and thus preventing outfitting.

Similar to the elastic boundary test, we have compares the best accuracy obtained with three different energy budgets. Figure 5.11 shows, using Cifar 10 data-set we observe a significant improvement in the accuracy in all cases using this optimisation. We observe about 6% improvement with 200 Joules of budget, 4% improvement in 300 Joules budget and about 2% improvement in 400 joules budget. We also observe more significant trend that this technique becomes more effective as the energy constraint become tighter and smaller.

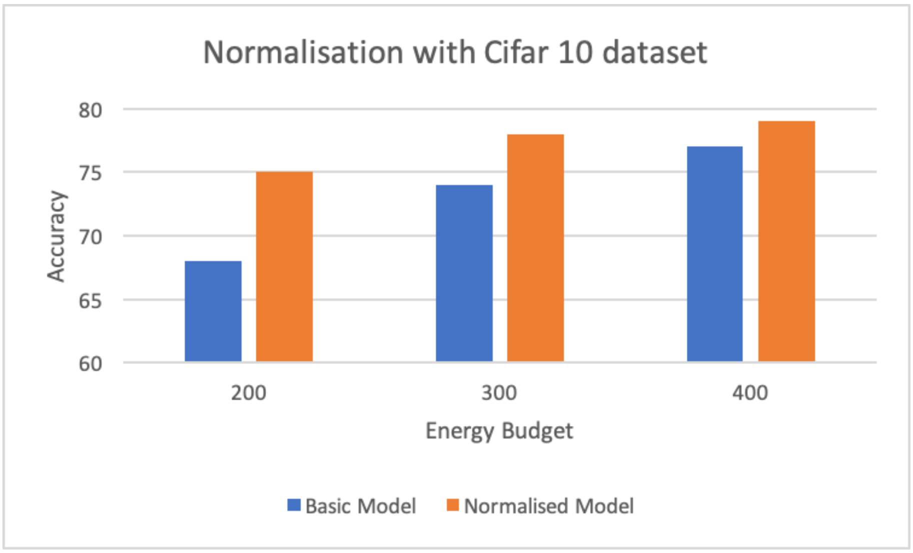


Figure 5.11: Comparing accuracy of Cifar 10 Database with Normalisation

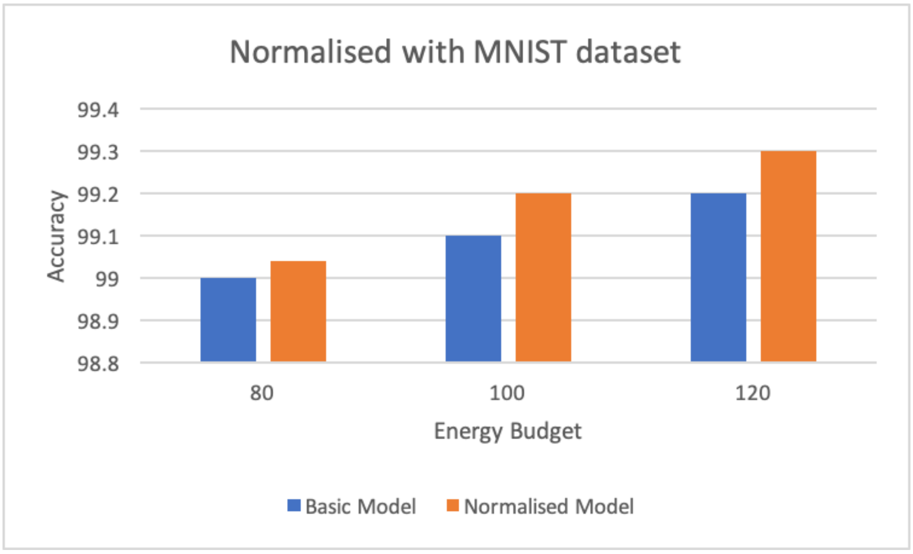


Figure 5.12: Comparing accuracy of MNIST Database with Normalisation

Figure 5.12 shows that we do also observe improvement in MNIST dataset; however, these improvements are much smaller due to the already high accuracy of the base model. We do observe 0.04%, 0.1% and 0.1% improvement for the energy budget of 80, 100 and 120 respectively.

### 5.4.5 Dynamic Model results

Finally, we apply the dynamic model and allow the Bayesian algorithm to choose number of layers along with other parameters. Figure 5.14 shows an improvement of 1%, 1.2% and 1.9% for the budgets of 200, 300 and 400 respectively. While in Figure 5.13, we observe no tangible benefit as the model is already at 99%.

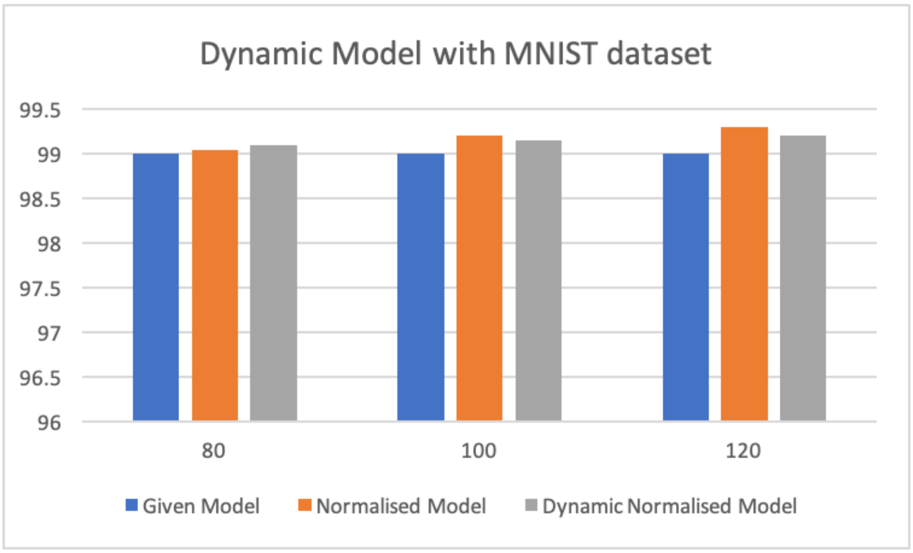


Figure 5.13: Comparing accuracy of Cifar 10 Database with Normalisation

However, if we compare the results with the given model, the dynamic models can provide similar accuracy as the given model with a smaller energy budget. We observe a 29% reduction in energy consumption in MNIST dataset with a 0.2% increase in the accuracy. Similarly, we are able to achieve a 2% improvement in the accuracy with a 34% reduction in energy consumption in Cifar10 dataset.

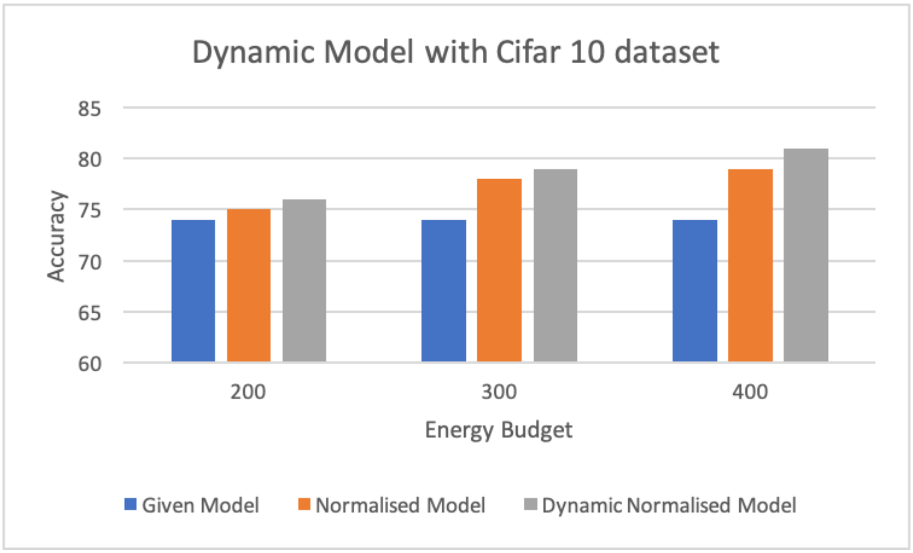


Figure 5.14: Comparing accuracy of MNIST Database with Normalisation

To conclude, in this chapter, we successfully developed an algorithm to generate a Neural Network with maximum accuracy given energy and parameter constraints. We started with a basic design to optimise hyper-parameters in the Neural Networks but developed it further to support optimisations such as creating dynamic models, elastic boundary, etc. We were able to reduce the energy consumption in the Final model of MNIST dataset by 29% and in Cifar10 dataset by 34% with minor gains in the accuracy of the network. In the next chapter, we discuss some further possible optimisations to the current design.

Chapter 6

# Conclusion and Future Work

## 6.1 Conclusion

In conclusion, as Neural Networks become more capable of solving ever-increasing challenging problems, they are bound to be more energy-consuming in the future. Therefore it is essential to develop techniques to conserve energy usage of these networks to balance their environmental impact and make them usable in battery-powered devices.

We explored different methods that could be employed to create an energy-efficient Neural Network classifier and found that the cascading model approach to be particularly useful as it doesn't compromise on the accuracy of the Neural Network and provides substantial energy savings without any specialised hardware. We observed that there are two types of Neural Network in this method – the Initial model which aims to classify input cheaply and the Final model, which aims to be accurate in its classification.

In this project, we looked to optimise both these type of Neural Networks. We proposed using semantic data such as Colour, Edges, Texture and Corners from the input images to help classify these images in an energy-efficient manner. We also looked for systematic techniques such as PCA and LDA for reducing the dimensions of the input space; therefore creating much simpler Neural Networks.

Moreover, we also developed a novel algorithm to arrange these semantic and systematic data classifiers expertly. While designing this algorithm, we took into consideration the advantages and the shortcoming of these semantic classifiers. We were able to reduce the energy consumption by 13% when comparing to the Scalable effort classifiers algorithm and 35% when compared to the Keras implementation for the Cifar 10 dataset. We were able to achieve this energy reduction without any loss in the accuracy of the network or using any specialised hardware.

Finally, we looked to optimise the energy consumption of the Final model. We developed an algorithm which employed Bayesian optimisation to choose the appropriate parameter for the highest possible accuracy given external energy constraints. In our approach, we restricted from making any significant assumption about the network and allowed the optimisation function to have greater flexibility in a network design such as choosing the number of layers in the network. Using this technique, we were able to achieve 29% and 34% energy reduction for MNIST and Cifar10 datasets respectively, without any loss in the accuracy of the network.

From the above project, we can conclude that we have made some considerable improvements in the current design of cascading networks. We are hopeful that after this contribution, they are even more widely used, and in turn, makes the evaluation of the Neural Networks more energy-efficient. Now we would discuss some more potential improvements to our implementation.

## 6.2 Future Work

In this section, we would discuss some of the potential improvement in the current work in this project.

### 6.2.1 Exploring Non-Visual data semantics

In our project, we have primarily explored semantics for image/video data sets. We can theoretically apply the same algorithms to non-visual data-sets provided that we can retrieve the semantic information for that type of data.

One such example could be audio data. In 'Features for audio and music classification' [79], it is shown that they can classify five general audio classes and seven popular music genres using audio features. The feature sets include low-level signal properties, mel-frequency spectral coefficients, and two new sets based on perceptual models of hearing. It is shown that classification is better, on average, if based on features from models of auditory perception rather than on standard features [79].

### 6.2.2 Safe Exploration for Gaussian Optimisation

In our implementation of the Bayesian optimisation, we had an eternal energy constraint that must be met to get a valid model result. We removed this constraint from the acquisition function where it was traditionally used to the objective function to prevent any simulation of the constraint. However, another solution could be to modify the Gaussian process prior to account for this constraint. This would be a similar approach that has been taken Safe Exploration for Optimisation with Gaussian Processes [23].

This technique uses the underline assumption in the Gaussian functions that similar decisions are associated with similar rewards. Thus it aims to balance exploration (learning about the function) and exploitation (identifying near-optimal decisions), while additionally ensuring safety throughout the process.

It models the unknown function from a Gaussian process (GP), and uses the predictive uncertainty to guide exploration of the future models. They establish the notion of "safely reachable" and uses the confidence bounds to assess the safety of as yet unexplored decisions [23].

Figure 6.1 describes the process more concretely. We can observe the algorithm starts with a random point in the required region. It then explores several points and tries and map out the objective function. Once it has some information about the function, it assesses the next points only if it is 'safely reachable' on the based on the current model.

This optimisation actively discards the useless points during the training and uses the model information to explore the regions, which would be optimisation on the current approach where the decision to explore the next point is solely taken based on the acquisition function.

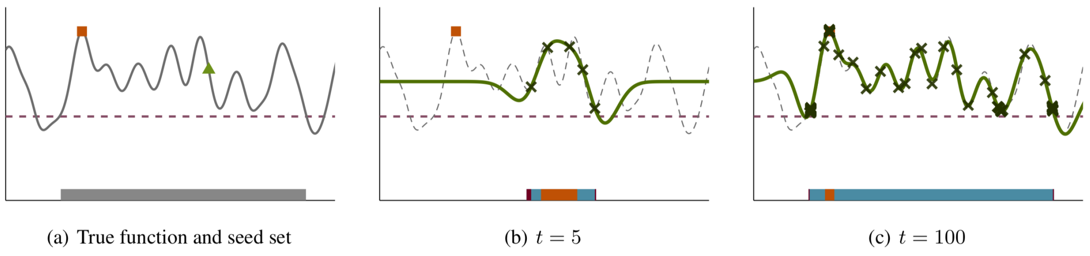


Figure 6.1: (a) The solid curve is the (unknown) function to optimise; the straight dashed line represents the threshold. In (b,c) The solid line is the estimated GP mean function after a number of observations shown as crosses [23]

.

There is no standard method to handle discrete input spaces in Bayesian optimisation. In our implementation, we have used rounding the values to the nearest integer to get the approximate value of the acquisition function. Although this solution might seem workable for a practical problem, it causes a discrepancy between what the acquisition function of the BO recommends as a next evaluation point and therefore, the final point evaluated.

### 6.2.3 Discrete search input spaces

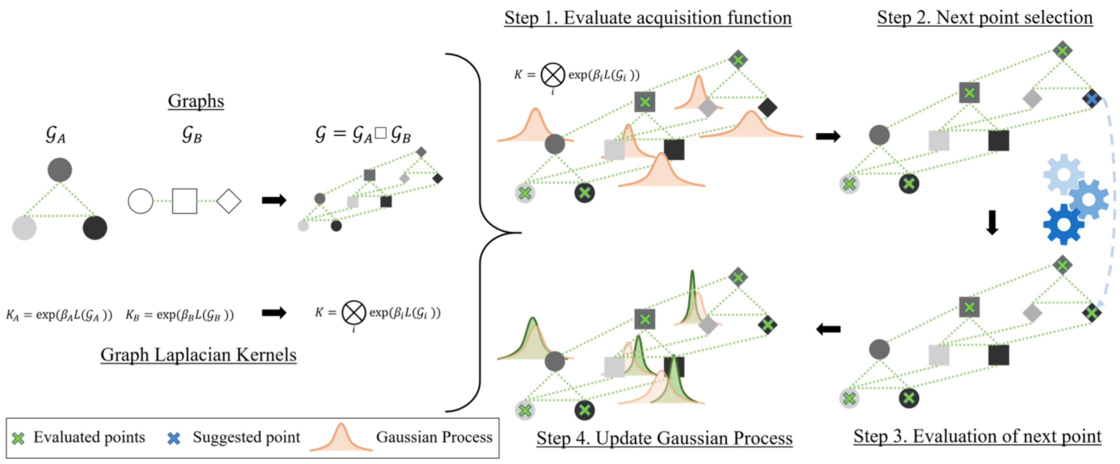

Step 1. Evaluate acquisition function
Step 2. Next point selection
Graphs
$\mathcal{G}_A$
$\mathcal{G}_B$
$\mathcal{G} = \mathcal{G}_A \square \mathcal{G}_B$
$K = \bigotimes_i \exp(\beta_i L(\mathcal{G}_i))$
$K_A = \exp(\beta_A L(\mathcal{G}_A))$
$K_B = \exp(\beta_B L(\mathcal{G}_B))$
Graph Laplacian Kernels
Evaluated points
Suggested point
Gaussian Process
Step 4. Update Gaussian Process
Step 3. Evaluation of next point

Figure 6.2: On the left we observe the graph Cartesian product with corresponding multiple kernels. On right we observe the evaluation of Bayesian optimisation using greedy local search [24]

Another limitation with the current approach is that it leads to flat values for the objective function in between the discrete inputs. This flatness is often ignored in the Gaussian Process Regressor and acquisition function when calculating the next iteration [80].

To counter these drawbacks, Combinatorial Bayesian Optimisation using Graph Representations [24] proposes an interesting solution. The basic intuition behind this solution is that as the size of the discrete spaces increases exponentially with respect to the number of variables; computation of the kernel becomes continuous. To achieve this, the kernel is expressed in sub-graphs. This division is done in linear time to make it possible to scale up for more complex problems.

Figure 6.2 explains the overview of the process. On the left, we can visualise how the problem with discrete parameters (using diffused kernel) is expressed as the graph Cartesian product of smaller sub-graphs. This diffused kernel can be adapted to account for individual tuning parameters per categorical variable, making it more flexible. While on the right, we can observe how this diffused kernel goes through the process of Bayesian optimisation discussed in Section 2.7.

# Appendix A

# Using Scripts

As part of the project we developed python scripts perform optimisations. There are three main folders in the project. 'Current_Models' is to train and evaluate the Keras and scalable effort classifiers neural networks. 'Semantic_Data' contains scripts for Chapter 4 of the report and 'Bayesian_Optimisation' contains scripts for Chapter 5 of the report.

## A.1 Using lower dimensional input data

First, setup and install the dependent python libraries. Tools such has 'pip' can be used to acheive. The main libraries are Keras, Tensorflow, CV2, s-tui, Numpy and Sklearn. Then you can run the scripts with the source code given in 'Semantic_Data' folder.

To run the scripts simply run 'python script_name argument' in terminal. The scripts currently uses the Cifar 10 dataset from Keras. Follow the table A.1 for more information on available scripts.

## A.2 Energy constraint Bayesian Optimisation

For this part, the dependant libraries are Keras, Tensorflow, s-tui, bayes_opt and Numpy. You will find the source code for this part in 'Bayesian_Optimisation' folder.

Similarity, to run the scripts we can you 'python script_name argument' in terminal. The scripts currently uses the Cifar 10 and MNIST datasets from Keras. Follow the table A.2 for more information on available scripts.

| **Script Name** | **Arguments** | **Description** |
|---|---|---|
| Corner_energy_measurement.py | None | Measure the energy usage to extract corners semantics |
| Corner_initial_model.py | Harris_free_variable | Extract corners for the corresponding value and create NN |
| Edge_energy_measurement.py | None | Measure the energy usage to extract edge semantics |
| Edge_initial_model.py | type_of_edge | Extract edge for the corresponding type and create NN |
| Gabor_energy_measurement.py | None | Measure the energy usage to extract texture semantics |
| Gabor_initial_model.py | texture_angle | Extract texture for the corresponding angle and create NN |
| HSV_energy_measurement.py | None | Measure the energy usage to extract colour semantics |
| HSV_initial_model.py | lower_limit, upper_limit | Extract colour between the corresponding limits and create NN |
| LDA_energy_measurement.py | None | Measure the energy usage for reducing input dimensions |
| LDA_initial_model.py | output_dimensions | Transform the data to the required dimension and create NN |
| Initial_design.py | None | Create initial design model using the semantics data from semantic_initial_value.txt |
| Optimised_design.py | None | Create optimised design model using the semantics data from semantic_initial_value.txt |

Table A.1: Scripts for lower dimensional input data

| **Script Name** | **Arguments** | **Description** |
|---|---|---|
| Cifar_10_sample.py | None | Simply train the neural network with known parameters for Cifar10 |
| Cifar_10_initial_model.py | energy_budget | Find the optimal non-dynamic model for Cifar10 |
| Cifar_10_normalised_model.py | energy_budget | Find the optimal normalised non-dynamic model for Cifar10 |
| Cifar_10_model_explore.py | energy_budget | Find the optimal normalised dynamic model for Cifar10 |
| MNIST_sample.py | None | Simply train the neural network with known parameters for MNIST |
| MNIST_initial_model.py | energy_budget | Find the optimal non-dynamic model for MNIST |
| MNIST_normalised_model.py | energy_budget | Find the optimal normalised non-dynamic model for MNIST |
| MNIST_model_explore.py | energy_budget | Find the optimal normalised dynamic model for MNIST |

Table A.2: Scripts for Energy constraint Bayesian Optimisation

# Bibliography

[1] A. Andrae and T. Edler, "On global electricity usage of communication technology: trends to 2030," *Challenges*, vol. 6, no. 1, pp. 117–157, 2015.

[2] A. Almeida, "Said the sky releases his first album with great voices." http://widefuture.com/2018/07/23/said-sky-releases-first-album-great-voices, 2018. Accessed on 15/01/2019.

[3] I. Strauss, "Here's what face you search for when you look into a crowd | from the grapevine." https://www.fromthegrapevine.com/innovation/heres-what-face-you-search-when-you-look-crowd. (Accessed on 05/27/2019).

[4] F.-F. Li, A. Karpathy, and J. Johnson, "Stanford cs class cs231n: Convolutional neural networks for visual recognition." http://cs231n.stanford.edu/, 2015. Accessed on 20/01/2019.

[5] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[6] D. L. Eliot, "Support vector machines (svm) for ai self-driving cars - ai trends." https://www.aitrends.com/ai-insider/support-vector-machines-svm-ai-self-driving-cars/. (Accessed on 05/27/2019).

[7] D. S. Sayad, "Support vector regression." https://www.saedsayad.com/support_vector_machine_reg.htm. (Accessed on 05/27/2019).

[8] K. D. Souza, "Powerkap - a tool for improving energy transparency for software developers on gnu/linux (x86) platforms." https://www.imperial.ac.uk/media/imperial-college/faculty-of-engineering/computing/public/1617-ug-projects/Krish-De-Souza---PowerKap;-A-tool-for-Improving-Energy-Transparency-for-Software-Developers-on-GNU.Linux-(x86)-platforms.pdf, 2017. Accessed on 17/01/2019.

[9] M. Lanaro, "Use tensorflow dnnclassifier estimator to classify mnist dataset." https://codeburst.io/use-tensorflow-dnnclassifier-estimator-to-classify-mnist-dataset-a7222bf9f940. (Accessed on 05/27/2019).

[10] J. Zürn, "Training a cifar-10 classifier in the cloud using tensorflow and google colab." https://medium.com/@jannik.zuern/training-a-cifar-10-classifier-in-the-cloud-using-tensorflow-and-google-colab-f3a5fbdfe24d. (Accessed on 06/14/2019).

[11] D. Burke, D. Jenkus, I. Qiqieh, S. Das, R. Shafik, and A. Yakovlev, "Significance-driven adaptive approximate computing for energy-efficient image processing applications," in *12th International Conference on Hardware/Software Codesign and System Synthesis: Special Session*, Newcastle University, 2017.

[12] S. Venkataramani, A. Raghunathan, J. Liu, and M. Shoaib, "Scalable-effort classifiers for energy-efficient machine learning," in *Proceedings of the 52nd Annual Design Automation Conference*, p. 67, ACM, 2015.

[13] P. Panda, A. Sengupta, and K. Roy, "Conditional deep learning for energy-efficient and enhanced pattern recognition," in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 475–480, IEEE, 2016.

[14] T.-J. Yang, Y.-H. Chen, and V. Sze, "Designing energy- efficient convolutional neural networks using energy-aware pruning.," *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[15] S. Raschka, "Linear discriminant analysis." https://sebastianraschka.com/Articles/2014_python_lda.html. (Accessed on 06/12/2019).

[16] T.-W. Chen, Y.-L. Chen, and S.-Y. Chien, "Fast image segmentation based on k-means clustering with histograms in hsv color space," in *2008 IEEE 10th Workshop on Multimedia Signal Processing*, pp. 322–325, IEEE, 2008.

[17] P. I. Frazier, "A tutorial on bayesian optimization," *arXiv preprint arXiv:1807.02811*, 2018.

[18] "efg's hsv lab report." http://www.efg2.com/Lab/Graphics/Colors/HSV.htm. (Accessed on 05/19/2019).

[19] J. H. Bear, "What is the hsv (hue, saturation, value) color model?." https://www.lifewire.com/what-is-hsv-in-design-1078068. (Accessed on 05/20/2019).

[20] "Hue, value, saturation | learn.." http://learn.leighcotnoir.com/artspeak/elements-color/hue-value-saturation/. (Accessed on 05/20/2019).

[21] A. Shah, "Through the eyes of gabor filter – anuj shah (exploring neurons)." https://medium.com/@anuj_shah/through-the-eyes-of-gabor-filter-17d1fdb3ac97. (Accessed on 05/20/2019).

[22] A. MK, "Linear discriminant analysis (lda) |." https://mlalgorithm.wordpress.com/2016/06/18/linear-discriminant-analysis-lda/. (Accessed on 06/13/2019).

[23] Y. Sui, A. Gotovos, J. Burdick, and A. Krause, "Safe exploration for optimization with gaussian processes," in *International Conference on Machine Learning*, pp. 997–1005, 2015.

[24] C. Oh, J. M. Tomczak, E. Gavves, and M. Welling, "Combinatorial bayesian optimization using graph representations," *arXiv preprint arXiv:1902.00448*, 2019.

[25] I. El Naqa and M. J. Murphy, "What is machine learning?," in *Machine Learning in Radiation Oncology*, pp. 3–11, Springer, 2015.

[26] E. García-Martín, "Energy efficiency in machine learning: A position paper," *In 30th Annual Workshop of the Swedish Artificial Intelligence Society SAIS 2017*, vol. 137, no. 3, pp. 68–72, 2017.

[27] I. Androutsopoulos, G. Paliouras, V. Karkaletsis, G. Sakkis, C. D. Spyropoulos, and P. Stamatopoulos, "Learning to filter spam e-mail: A comparison of a naive bayesian and a memory-based approach," *arXiv preprint cs/0009009*, 2000.

[28] M. J. Pazzani and D. Billsus, "Content-based recommendation systems," in *The adaptive web*, pp. 325–341, Springer, 2007.

[29] S.-L. Chang, L.-S. Chen, Y.-C. Chung, and S.-W. Chen, "Automatic license plate recognition," *IEEE transactions on intelligent transportation systems*, vol. 5, no. 1, pp. 42–53, 2004.

[30] S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back, "Face recognition: A convolutional neural-network approach," *IEEE transactions on neural networks*, vol. 8, no. 1, pp. 98–113, 1997.

[31] Y. Elovici, A. Shabtai, R. Moskovitch, G. Tahan, and C. Glezer, "Applying machine learning techniques for detection of malicious code in network traffic," in *Annual Conference on Artificial Intelligence*, pp. 44–50, Springer, 2007.

[32] A. Coleman, "How much does it cost to keep your computer online? (lots, it turns out)." http://www.telegraph.co.uk/business/energy-efficiency/cost-keeping-computer-online/, 2017. Accessed on 15/01/2019.

[33] E. Fryer, "Data centres and power: Fact or fiction?." https://www.techuk.org/images/programmes/DataCentres/Data_Centres_and_Power.pdf. Accessed on 15/01/2019.

[34] A. S. Andrae, "Total consumer power consumption forecast," *gehalten auf der Nordic Digital Business SummitHelsinki, Finland, 2017*, 2017.

[35] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[36] D. Li, X. Chen, M. Becchi, and Z. Zong, "Evaluating the energy efficiency of deep convolutional neural networks on cpus and gpus," in *2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom)(BDCloud-SocialCom-SustainCom)*, pp. 477–484, IEEE, 2016.

[37] S. Leroux, S. Bohez, E. De Coninck, T. Verbelen, B. Vankeirsbilck, P. Simoens, and B. Dhoedt, "The cascading neural network: building the internet of smart things," *Knowledge and Information Systems*, vol. 52, no. 3, pp. 791–814, 2017.

[38] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[39] L. Deng, J. Li, J.-T. Huang, K. Yao, D. Yu, F. Seide, M. Seltzer, G. Zweig, X. He, J. Williams, *et al.*, "Recent advances in deep learning for speech research at microsoft," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 8604–8608, IEEE, 2013.

[40] A. Karpathy, "Cs231n: Convolutional neural networks for visual recognition." http://cs231n.github.io/convolutional-networks/, 2018. Accessed on 20/01/2019.

[41] R. Berwick and V. Idiot, "An idiot's guide to support vector machines (svms)." http://web.mit.edu/6.034/wwwbob/svm-notes-long-08.pdf. Accessed on 23/01/2019.

[42] "204.6.8 svm : Advantages disadvantages and applications – statinfer." https://statinfer.com/204-6-8-svm-advantages-disadvantages-applications/. (Accessed on 05/27/2019).

[43] "Powerdef. oxford power definition." https://en.oxforddictionaries.com/definition/power, 2018. Accessed on 15/01/2019.

[44] P. K. Mike Yi, "Intel power gadget." https://software.intel.com/en-us/articles/intel-power-gadget-20, 2018. Accessed on 17/01/2019.

[45] "psutil documentation — psutil 5.6.3 documentation." https://psutil.readthedocs.io/en/latest/. (Accessed on 06/10/2019).

[46] "Github - amanusk/s-tui: Terminal-based cpu stress and monitoring utility." https://github.com/amanusk/s-tui. (Accessed on 06/10/2019).

[47] Y. LeCun, C. Cortes, and C. J. Burges, "The mnist database." http://yann.lecun.com/exdb/mnist/, 2018. Accessed on 18/01/2019.

[48] A. Krizhevsky, "The cifar-10 dataset." https://www.cs.toronto.edu/~kriz/cifar.html, 2018. Accessed on 18/01/2019.

[49] L. Fei-Fei, R. Fergus, and P. Perona, "Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories," *Computer vision and Image understanding*, vol. 106, no. 1, pp. 59–70, 2007.

[50] J. Zhu, Z. Qian, and C.-Y. Tsui, "Lradnn: High-throughput and energy-efficient deep neural network accelerator using low rank approximation," *IEEE Asia and South Pacific Design Automation Conference*, pp. 581–586, 2016.

[51] Y.-H. Chen, J. Emer, and V. Sze, "Using dataflow to optimize energy efficiency of deep neural network accelerators," *IEEE Micro*, vol. 37, no. 3, pp. 12–21, 2017.

[52] M. Imani, M. Samragh, Y. Kim, S. Gupta, F. Koushanfar, and T. Rosing, "Rapidnn: In-memory deep neural network acceleration framework," *arXiv preprint arXiv:1806.05794*, 2018.

[53] K. Kim, J. Kim, J. Yu, J. Seo, J. Lee, and K. Choi, "Dynamic energy-accuracy trade-off using stochastic computing in deep neural networks," in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2016.

[54] I. Qiqieh, R. Shafik, G. Tarawneh, D. Sokolov, and A. Yakovlev, "Energy-efficient approximate multiplier design using bit significance-driven logic compression," in *Proceedings of the Conference on Design, Automation & Test in Europe*, pp. 7–12, European Design and Automation Association, 2017.

[55] S. Teerapittayanon, B. McDanel, and H. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pp. 328–339, IEEE, 2017.

[56] T. Bolukbasi, J. Wang, O. Dekel, and V. Saligrama, "Adaptive neural networks for efficient inference," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 527–536, JMLR. org, 2017.

[57] E. Park, D. Kim, S. Kim, Y.-D. Kim, G. Kim, S. Yoon, and S. Yoo, "Big/little deep neural network for ultra low power inference," in *Proceedings of the 10th International Conference on Hardware/Software Codesign and System Synthesis*, CODES '15, (Piscataway, NJ, USA), pp. 124–132, IEEE Press, 2015.

[58] D. Roy, P. Panda, and K. Roy, "Tree-cnn: a hierarchical deep convolutional neural network for incremental learning," *arXiv preprint arXiv:1802.05800*, 2018.

[59] P. Panda, A. Ankit, P. Wijesinghe, and K. Roy, "Falcon: Feature driven selective classification for energy-efficient image recognition," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 12, 2017.

[60] S. M. T. S. Zadeh, A. Mehrsina, and M. Basirat, "An accelerated k-means clustering algorithm for image segmentation.," *Journal of Theoretical & Applied Information Technology*, vol. 48, no. 3, 2013.

[61] P. Panda, S. Venkataramani, A. Sengupta, A. Raghunathan, and K. Roy, "Energy-efficient object detection using semantic decomposition," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 9, pp. 2673–2677, 2017.

[62] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in neural information processing systems*, pp. 2951–2959, 2012.

[63] C. A. Shoemaker, R. G. Regis, and R. C. Fleming, "Watershed calibration using multistart local optimization and evolutionary optimization with radial basis function approximation," *Hydrological sciences journal*, vol. 52, no. 3, pp. 450–465, 2007.

[64] E. Brochu, V. M. Cora, and N. De Freitas, "A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning," *arXiv preprint arXiv:1012.2599*, 2010.

[65] "Github - fmfn/bayesianoptimization: A python implementation of global optimization with gaussian processes.." https://github.com/fmfn/BayesianOptimization. (Accessed on 06/09/2019).

[66] "sklearn.gaussian_process.kernels.matern — scikit-learn 0.21.2 documentation." https://scikit-learn.org/stable/modules/generated/sklearn.gaussian_process.kernels.Matern.html. (Accessed on 06/10/2019).

[67] "keras/examples at master · keras-team/keras · github." https://github.com/keras-team/keras/tree/master/examples. (Accessed on 06/10/2019).

[68] P. Veličković, D. Wang, N. D. Lane, and P. Liò, "X-cnn: Cross-modal convolutional neural networks for sparse datasets," in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 1–8, IEEE, 2016.

[69] W. Fleshman, E. Raff, J. Sylvester, S. Forsyth, and M. McLean, "Non-negative networks against adversarial attacks," *arXiv preprint arXiv:1806.06108*, 2018.

[70] J. Aigrain and M. Detyniecki, "Detecting adversarial examples and other misclassifications in neural networks by introspection," *arXiv preprint arXiv:1905.09186*, 2019.

[71] "Understanding gabor filters | perpetual enigma." https://prateekvjoshi.com/2014/04/26/understanding-gabor-filters/. (Accessed on 05/20/2019).

[72] W. Gao, X. Zhang, L. Yang, and H. Liu, "An improved sobel edge detection," in *2010 3rd International Conference on Computer Science and Information Technology*, vol. 5, pp. 67–71, IEEE, 2010.

[73] S. J. Bedros, "Edge detection." http://www.me.umn.edu/courses/me5286/vision/VisionNotes/2017/ME5286-Lecture7-2017-EdgeDetection2.pdf. (Accessed on 06/11/2019).

[74] C. G. Harris, M. Stephens, *et al.*, "A combined corner and edge detector.," in *Alvey vision conference*, vol. 15, pp. 10–5244, Citeseer, 1988.

[75] D. Stamoulis, E. Cai, D.-C. Juan, and D. Marculescu, "Hyperpower: Power-and memory-constrained hyper-parameter optimization for neural networks," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 19–24, IEEE, 2018.

[76] M. T. Morar, J. Knowles, and S. Sampaio, "Initialization of bayesian optimization viewed as part of a larger algorithm portfolio." `http://ds-o.org/images/Workshop_papers/Morar.pdf`, 2017. (Accessed on 06/09/2019).

[77] E. C. Garrido-Merchán and D. Hernández-Lobato, "Dealing with integer-valued variables in bayesian optimization with gaussian processes," *arXiv preprint arXiv:1706.03673*, 2017.

[78] A. Koutsoukas, K. J. Monaghan, X. Li, and J. Huan, "Deep-learning: investigating deep neural networks hyper-parameters and comparison of performance to shallow methods for modeling bioactivity data," *Journal of cheminformatics*, vol. 9, no. 1, p. 42, 2017.

[79] M. McKinney and J. Breebaart, "Features for audio and music classification," p. pages 151–158, Proc. Int. Symp. Music Inf. Retrieval (ISMIR'2003), (Baltimore, USA), October 2003.

[80] E. C. Garrido-Merchán and D. Hernández-Lobato, "Dealing with categorical and integer-valued variables in bayesian optimization with gaussian processes," *arXiv preprint arXiv:1805.03463*, 2018.