POLITECNICO DI MILANO

COMO CAMPUS

Master of Science

In

COMPUTER SCIENCE AND ENGINEERING

PER LAYER TIME COMPUTATION OF CNN TRAINING WITH TENSORFLOW

Supervisor:

Prof. Danilo Ardagna – Politecnico Di Milano

Co-Supervisor:

Dr. Marco Lattuada – Politecnico di Milano

Department of Electronics, Information and Bioengineering

Master Graduation Thesis By

Muhammad Imran

850749/10517583

Academic year 2017/2018

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to my supervisors **Professor Danilo Ardagna** and Dr **Marco Lattuada** for giving me this opportunity to work with them on this project and for their guidance and valuable feedback along the whole process of this thesis.

I would like to express my gratitude to Politecnico Di Milano for giving me this opportunity to study here and enabling me to work with people from different cultures and backgrounds and helped me better myself not only academically but helping me learn a lot about life in this short period.

I would also like to thank all my professors form whom I have learned a lot.

Last but not the least, I express my gratitude to my family for being supportive and inspiring me to keep going at times when it seemed difficult to achieve my goals.

1 Contents

List of Figures

List of tables

Abstract

This thesis is studying the basic concepts of Convolutional Neural Networks. Influence of different configuration settings on the ability of the network to train, more specifically the effect of these different configurations on the inner layers computation time of a model. We will study an approach to collect the per layer execution time of training a CNN. The results of running the training process with different configurations will be discussed.

Sommario

Questa tesi sta studiando i concetti di base delle Reti Neurali Convolturali. Influenza di diverse impostazioni di configurazione sulla capacità della rete di addestrare, più specificamente l'effetto di queste diverse configurazioni sul tempo di calcolo degli strati interni di un modello. Studieremo un approccio per raccogliere il tempo di esecuzione per strato dell'addestramento di una CNN. Verranno discussi i risultati dell'esecuzione del processo di formazione con diverse configurazioni.

Chapter 1

1 Introduction

Machine learning and Deep Learning are the topics which are paid huge attention since the last several years. This is very interesting in the sense that research in artificial intelligence started back in the mid-20th century with bold promises which were not materialized by the end of the century. It was not that the research didn't bring any fruitful innovations, but failure was on the part that to make machines see and understand. One of the many reasons for this was the fact that vision and voice recognition are much more complex process.

The breakthrough was made ten years later in the application of neural networks. This was somewhat unexpected because of the reason that most of the research community didn't pay attention to it anymore. Neural Networks are one of the oldest models which was perceived to be a failed endeavor. Since 2010, technical news sources are reporting stories of deep learning application in almost every aspect of life. Since then companies like Amazon, Apple, Facebook or Google from Fortune 500 Companies are pouring huge resources into machine learning research.

These developments in this field has highly influenced the current technologies in use, from mobile phones in our pockets to identify human faces, understanding and processing voice commands to react in appropriate way to produce the results. To autonomous vehicles which are already commercialized by some of the big companies like Tesla.

Among all these contributions in the field of AI the biggest one most likely is the contribution of Convolutional Neural Networks (CNN). Aim of the thesis is to document its background and basic theoretical concepts, implementing several CNN models in one of the available software tools and collecting the per layer information of the CNN models during training process. we will test two models on some classes from Large Scale Visual Recognition Challenge (ILSVRC) otherwise known as ImageNet dataset on a cloud machine with enough resource power to handle the demands of these deep learning networks. The results will be visualized with respect to the computation time of different layers of this model.

Chapter 2

2 State of the Art

This chapter aims to provide a description of the fields and technologies relative to this thesis.
Section 2.1 discusses Cloud computing; next Section 2.2 describes Machine learning and TensorFlow deep learning framework and an introduction into GPUs and deep learning.

2.1 Cloud Computing

The last decade has been very important in terms of cloud computing, people realized that information can be processed more efficiently centrally, on large farms of computing and storage systems accessible via the internet. In the computation process the resources used are the ones in distant data centers rather than the ones of local computing systems, this approach is network-centric computing and network-centric content. This advancements in networking and other areas made it possible and the acceptance of two new computing models, Grid Computing in 1990s, and utility computing and cloud computing since 2005. Utility computing concentrates hardware and software resources in a large data centers and users can pay for the service as they use computing, storage, and communication resources. Cloud infrastructure is often required for utility computing, but the focus is on the business model for providing the computing services. Cloud computing is a path to utility computing embraced by major IT companies such as Amazon, Apple, Google, HP, IBM, Microsoft, Oracle, and others [1].

- Cloud computing provides Elastic Services by use of Internet technologies. The ability to dynamically acquire computing resources and support a variable workload is known as elastic computing. Cloud service provider maintains a massive infrastructure to support elastic services.
- The service provided to the user is metered and the user is charged for the resources that are used.
- Maintenance and security are ensured by service providers.
- Economy of scale allows service providers to operate more efficiently due to specialization and centralization.

- Resource Multiplexing makes Cloud computing cost-effective, these lower costs for service providers are passed on to the cloud users, making it cost-effective.
- The application deployed on cloud, its data is stored closer to the site where it is being used in a device- and location- independent manner, this strategy of data storage increases reliability and security, and lowers communication costs at the same time.

Cloud computing can be applied in different scientific and engineering applications, computational financing, gaming, data mining and social networking as well as many more data-intensive and computational tasks. Cloud enables to store extensive range of data, it can include enterprise management data to personal data such as photos, videos, and other files [1].
The following sections 2.1.1 describes the fundamentals of cloud computing and section 2.1.2 describes the Cloud Computing architecture.

2.1.1 Fundamentals of Cloud Computing

Cloud computing model making use of internet, provides computing power, software, storage services and platforms to external customers on demand. The way cloud is changing computation is because of these few properties.

- A cloud is a combination of similar set software and hardware in a single administrative domain. In this order, security, fault tolerance, resource management and quality of service are less challenging compared to a different environment with resources in multiple administrative domains.
- The focus of cloud computing is on enterprise computing, while the adoption by industrial organizations, healthcare organizations, financial institutions and so on has a potentially large impact on the economy.
- A cloud on the consumer end gives the illusion of limitless computing resources; because of this elasticity application designers are not constrained to a single system.
- Cloud eliminates the need for up-front financial investment since it is based on a pay-as-you-go approach. This enables the cloud to attract new applications and new users for existing applications, fueling a new era of industrywide technological advancements.

Resources and services can be scaled up or down according to the needs or demand. Cloud computing providers typically charge on pay-per-use model. Instead of investing in expensive computing infrastructure or data centers, companies or customers can rent a service from storage to applications from the cloud provider and pay for the resources being used. The big advantage of pay-as-you-go method is that resources are not wasted, since only the resource used are charged, in comparison to allowing a certain amount of

resources that may or may not be used. In traditional enterprise architecture design, the focus is on architecting data storage to allow maximum workload, in public cloud, the pay-as-you-go method allows you to be charged only for what you store. Pay-as-you-go platforms, such as Amazon EC2, provides services by allowing users to design compute resources charges by what is used. Users make the choice of CPU, storage, security, memory, operating system, networking capacity and access controls, and any other software needed for the environment to run.

2.1.2 Cloud Computing Characteristics

2.1.2.1 Defining Cloud Computing

Cloud Computing in its simplest forms means storing and accessing data and programs over the internet from a remote location or computer instead of local computer. This approach of accessing from remote location has several properties such as scalability, elasticity etc., which is way considerably different a normal remote machine. Cloud is just a metaphor for the internet. When data is stored or run a program locally on a pc it's called local storage and computing, while in cloud computing, data or programs needs to be accessed via internet. The result is the same, however, with use of internet cloud computing can be done anywhere, anytime, and by any device.

The formal definition of cloud computing comes from the National Institute of Standards and Technology (NIST) [2]: "Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models [1].

Meaning computing resource or infrastructure, it might be server hardware, storage, network, or application software provided by cloud vendor or providers site, can be accessible over the internet from any remote location and any local computing device. In addition, the usage or access of the resources cost will be only the ones for which resources are used, known as pay-as-you-go model. If need arises, more computing resources are made available by the provider to the environment. Minimal Management effort implies that at the customer's side, the maintenance of computing systems is very marginal, as they will use their local computing device for accessing cloud-based resources, not for those computing resources managed at the provider's side. Details of five essential characteristics, three service models, and four deployment models are provided in the 5-4-3 principles in Section 2.1.2.2. Many vendors, pundits, and experts refer to NIST, and

both the International Standards Organization (ISO) and the Institute of Electrical and Electronics Engineers (IEEE) back the NIST definition.

2.1.2.2 Principles of Cloud computing

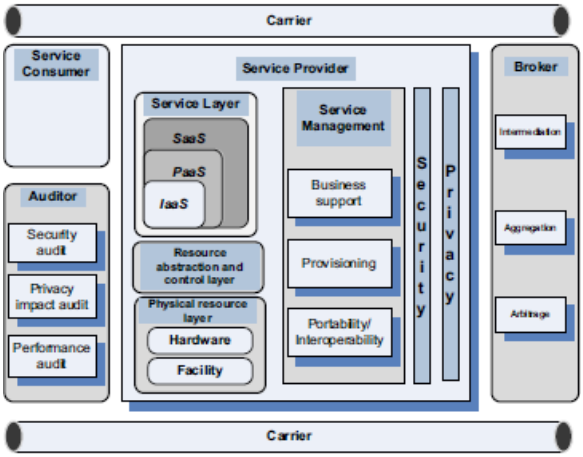

Figure 2-1 The entities involved in service-oriented computing and in cloud computing, according to NIST.[2]

NIST describes the 5-4-3 principles for cloud computing as follows

a) The five essential characteristic features that promote cloud computing
b) The four deployment models that are used to narrate the cloud computing opportunities for customers while looking at architectural models
c) The three important and basic service offering models of cloud computing.

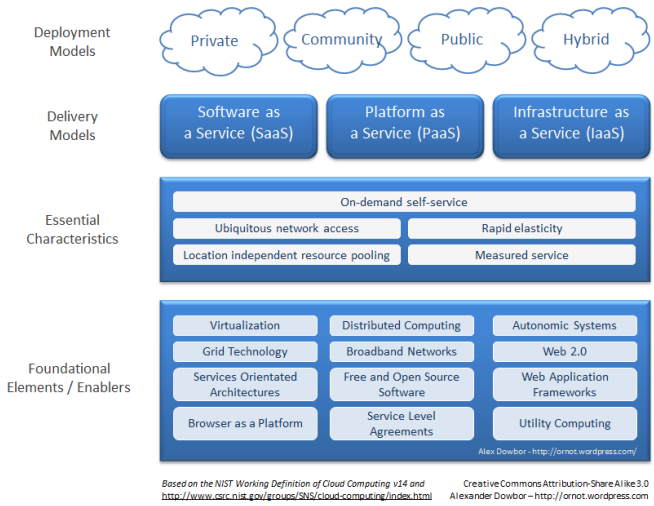

Figure 2-2 NIST 5-4-3 Cloud Depiction [3]

2.1.2.3 Five Essential Characteristics

Cloud computing has five essential characteristics, which are shown in Figure 2.3. Each characteristic is essential, which means that if any of these characteristics is missing, then it is not cloud computing [4]:

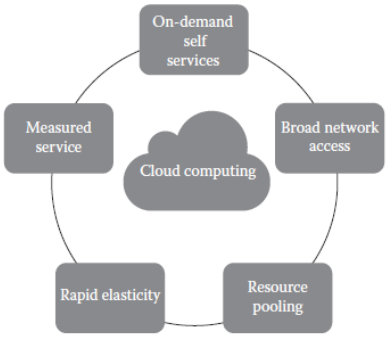

Figure 2-3 The essential characteristics of cloud computing. [4]

1. **On-demand self-service**: Cloud computing enables the provision of computing services to the consumer on demand. For example, network usage and server time can be provisioned automatically as the application requires it.
2. **Broad network access**: Cloud computing uses internet technologies to cater services and accessed using components that promote the usage of different thin or thick client platforms (i.e. mobile phones, laptops, etc.).
3. **Elastic resource pooling**: In Cloud computing the computing resources are pooled, the physical and virtual resources are allocated and deallocated dynamically to serve consumers using a multitenant model. This brings a kind of location independence, where the consumer has no knowledge of the exact location of provided services but can determine the location at higher level of abstraction (e.g., data center, state, country). The resources include storage, processing, memory and network bandwidth.
4. **Rapid elasticity**: Cloud computing provides resources to consumer, which can be rapidly and elastically provisioned, sometimes automatically to swiftly scale out and rapidly released to scale in. The capabilities often appear unlimited and can be bought in any quantity at any time.
5. **Measured Service**: The services provided by cloud are metered at some level based on the type of service (e.g., processing, bandwidth, storage), so the resources can be controlled and optimized automatically. Because of this metering capability, resource usage can be monitored, controlled and reported which gives transparency to both consumer and the service provider.

2.1.2.4 Four Cloud Deployment Models

Deployment models describe the ways with which the cloud services can be deployed or made available to its consumers, depending on the organizational structure and the provisioning location. To describe it further, cloud-based computing resources, the locations where the data and services are acquired and provisioned for the consumer can have various forms. Four deployment models are usually distinguished, namely, public, private, community, and hybrid cloud service usage.

1. **Private cloud:** A private cloud infrastructure is provisioned exclusively for a single organization with multiple consumers. It may be owned, managed and operated by the organization, a third party or combination of them. It can be on or off premises.
2. **Public cloud:** A public cloud infrastructure is provisioned for open use by the public. It may be owned, managed, and operated by a business, academic, or government organization, or some combination of them. It exists on the premises of the cloud provider.
3. **Community cloud:** A community cloud infrastructure is shared by several organizations and supports a specific community that has shared concerns (e.g., mission, security requirements, policy, and compliance considerations). It may be managed by the organizations or a third party and may exist on premise or off premise.
4. **Hybrid cloud:** The hybrid cloud infrastructure is a composition of two or more distinct cloud infrastructures (private, community, or public) that remain unique entities but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load balancing between clouds).

2.1.2.5 Three Service Offering Models

Cloud computing offers three kind of service offerings to the customer, they are Software as a Service (SaaS), Platform as a Service (Paas) and infrastructure as a service (IaaS). These service offerings are also known as service-platform-infrastructure (SPI) model of the cloud and can be seen in the figure 2.4. SaaS is a software distribution in which applications are hosted by a vendor or service provider and made available to customers over a network, typically internet. PaaS is a paradigm for delivering operating systems and associated services over the internet without downloads or installation. While IaaS comprises of outsourcing the equipment used to support operation, including storage, hardware, servers and networking components.

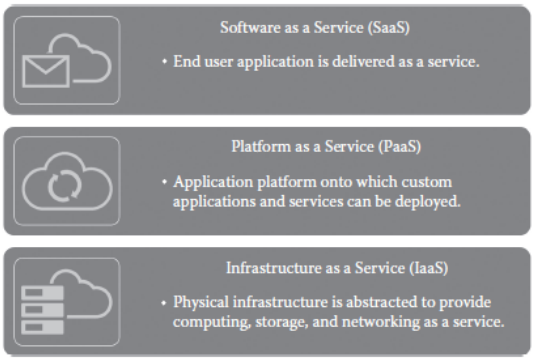

Figure 2-4 SPI—service offering model of the cloud. [4]

1. **Cloud SaaS:** Cloud SaaS enables the consumer to use the provider's applications running on a cloud infrastructure, which includes network, servers, operating systems, storage, and even individual application capabilities, except for limited user-specific application configuration settings. This provides the capability to access the application from various client devices through either a thin client interface, such as a web browser (e.g., web-based-e-mail), or a program interface. Controlling the underlying cloud infrastructure is not done by the consumer. Common example of service includes customer relationship management (CRM), business intelligence analytics, and online accounting software.
2. **Cloud PaaS:** Cloud PaaS enables consumer to deploy its own applications developed using programming languages, libraries, services, and tools supported by the provider on the cloud. In PaaS the management or control of the underlying cloud structure is not provided, but the control over the deployed applications and possibly configuration settings for the application-hosting environment. In short, it's a packaged and ready to use/run development or operating framework. PaaS vendor provides the networks, servers and storage and manages the levels of scalability and maintenance. The client usually pays for the services used. Some of the big PaaS providers include Google App Engineer and Microsoft Azure Services.
3. **Cloud IaaS:** Cloud IaaS enables the consumer to provision processing, storage, networks, and other fundamental computing resources on pay-per-use basis. Where the consumer can deploy and run arbitrary software, which may consist of operating systems and applications. Here the consumer does not manage or control the underlying cloud infrastructure but has control over the operating systems, storage and deployed applications and possibly limited control of select networking components (e.g., host firewalls). The equipment is owned by the service provider and is responsible for housing, cooling operation, and maintenance. Amazon Web Services (AWS) is a popular example of a large IaaS provider.

The major difference between PaaS and IaaS is the amount of control that users have. PaaS allows vendors to manage everything, while IaaS requires more management from the customer side.

2.1.3 Cloud Architecture

Architecture is the base for which any technological model functions, a hierarchical view describing the technology. The cloud architecture describes the way it works. It comprises of the dependencies needed, and the components that work on it. The Cloud is a relatively new technology and is completely reliant on internet technologies.

The architecture of cloud is a combination of components and subcomponents that are vital for cloud computing. Mostly these cooperative components consist of a front end, a backend platform, a cloud-based delivery system and a network. The frontend and backend are linked with a network, typically via internet by way of a delivery system.

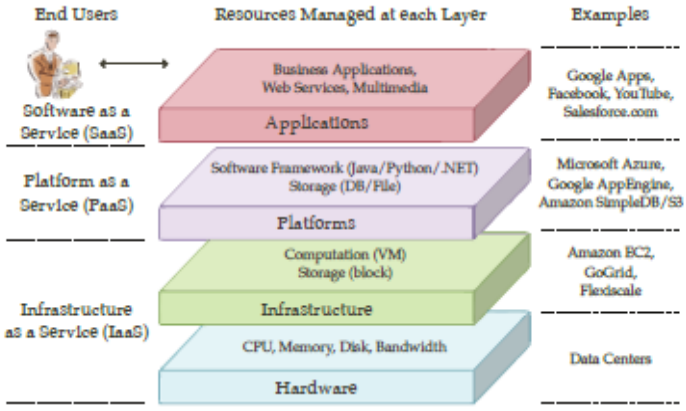

Figure 2-5 Cloud computing architecture. [5]

2.1.3.1 Architectural Layers of cloud computing

We can categorize the architecture of cloud computing into four layers, the physical layer, the infrastructure layer, the platform layer and the application layer, as described in figure 2.5.

- The physical layer deals with the physical assets of the cloud including routers, servers, switches, cooling systems and power.
- The infrastructure layer also called the virtualization layer is responsible for making a pool of storage capacity and computing resources by partitioning the physical resources using virtualization technologies such as KVM and VMware.
- The platform layer built on top of the infrastructure layer, comprising of the operating systems and requisition structures.
- The application layer is responsible for the actual cloud provisions, for e.g. Business Applications, Multimedia & web Services [4].

2.2 Machine Learning

Machine Learning is one of the domains of artificial intelligence (AI). The objective of machine learning is to understand the structure of data and find hidden correlation/patterns and fit that data into models that can be understood and utilized by people.

Even tough machine learning being a domain within computer science, it varies from traditional computational approaches. In traditional computing, algorithms are composed of programming instructions to calculate or solve a problem. While in machine learning algorithms a computer is trained on data inputs, and then applying statistical analysis to the output values and checking if it falls within a specific range. This characteristic of machine learning enables computers to build models from sample data and automate decision making process based on data input.

Currently, any user of technology has in one way or another benefitted from Machine learning, Facial recognition technology has allowed media platforms to automatically tagging of images. Optical character recognition (OCR) technology has helped convert images to editable text. Biggest of all in the consumer side, it has helped big companies like amazon, YouTube, google, Spotify by using recommendation engines powered by machine learning, to recommend movies, music, product more precisely. Autonomous vehicles navigation is made possible with Machine learning.

2.2.1 Machine Learning Methods

In machine learning, tasks are generally classified into broad categories. These categories are based on how learning is received or how feedback on the learning is given to the system developed.

Two of the most widely used/adopted machine learning methods are supervised learning which trains the model based on example input and output data that is labeled by humans, Unsupervised learning on the other hand, provides the alogorithm with no labeled data to allowing it find structure within its input data.

2.2.1.1 Supervised Learning:

Supervised Learning is the type of learning in which trained algorithms are fed labeled data. The algorithm is given data with correct outputs. The learning algorithm finds error and learns by comparing its actual value. Supervised learning follows the method of classification, regression, prediction and gradient boosting. The algorithm uses certain patterns to find the values of labels of unlabeled data. This helps predict future events based on historical data. Some examples of Supervised learning are Regression, Neural Networks, Decision Tree, Random Forest, KNN, Logistic Regression etc. Our focus will be on Neural Networks and specifically Convolutional Neural Networks.

2.2.1.2 Unsupervised Learning

Unsupervised learning, the type of learning in which the data is unlabeled, finding pattern and commonalities among the data is left for the learning algorithm. Since unlabeled data is available more than labeled data, the ML methods facilitating unsupervised learning are valuable. Unsupervised learning can be used to learn hidden patterns in a dataset, but can also be used for feature learning, allowing the machine to discover the representation needed for raw data automatically. Examples include, Apriori algorithm, k-means.

Apriori algorithm a popular algorithm for extracting frequent itemset. It generates association rules form a given data set. Association rules implies that if an item A occurs, then item B will also occur with a certain probability. Majority of association rules generated are of form if-then, e.g. if certain event happened, then following event also happened with a probability.

K-means is one of the popular clustering algorithms. This algorithm stores k centroids, using it define clusters. A point is considered to belong to certain cluster if the it is closer to the centroid of that cluster than any other cluster.

2.3 Artificial Neural Networks

Artificial neural network is a supervised machine learning model. When trained a model can learn a function from the labeled examples training examples with in which the output is specified. The inspiration for neural network comes from brain function and its ability to learn. Furthermore, the way biological neurons are computational units of the brain, a neural network uses neuron. The concept behind neural networks is, that a single neuron can only learn basic function, but millions of them combined can achieve complex tasks, such as recognizing different objects. [16]

Neurons are united in groups of layers, which are connected to each other. There are linked parameters also known as weights, among neurons of some layers, which can change the value of the input by an activation function, which calculates a value called activation. The parameters are subjected to change during the training process. The layers having the parameters are called trainable layers. An example of trainable layer is a dense layer that is connecting every input neuron with the neuron it holds. An example of non-trainable layer is pooling layer described below in section 3.2. In dense layers parameters are the number of neurons and the type of activation function. The number of parameter is dependent on the input layer, and the output layer is shaped for the expected output. The layers in between input and output layers are called hidden layers. The network architecture is acyclic, resulting in a directed acyclic graph.

2.3.1 Obtaining a prediction – forward-pass

The network takes the input as first layer. The input passes from layer to layer, every layer receives an input carry out activation function, retrieves the activation and passes it into

next layer until the output layer is reached which returns the result. This process is called forward pass, since the data is going from input to output layer.

2.3.2 Learning a function – backpropagation

The training is done in many iterations, in every iteration the network parameters are updated to improve performance. To train a network, it expects to be presented with example of input. Initially in the learning process the network has no idea how to treat input as the parameters are initiated to a small random number. the result of the forward pass is most likely wrong at this stage, to make correction the measure of error is defined, called loss or cost. The loss is resulted by evaluating loss function of desired output and prediction of the network. The learning algorithm needs to decide how to update all network parameters to decrease loss. Subsequently, the negative gradient of loss in respect to parameters is calculated by recursively applying chain rule layer by layer from output towards input. This process is called backpropagation. Backpropagation is repeated for every example, and the learning rate of the average of all obtained negative gradients is added to weights and update them. This results in optimizing all network parameters, and converges to local minima, called as stochastic gradient descent (SGD) [17] shown in equation 3.1, where w_{i+1} are new learned parameters, w_i are current parameters, α is learning rate, ∇w is gradient with respect to w_i and $L(w_i)$ is a loss function.

$$w_{i+1} = w_i - \alpha \nabla w_i L(w_i) \tag{3.1}$$

2.3.3 Optimizers

There are different variations of optimizers based on SGD, which try to overcome different shortcomings that SGD had when converging. Some of the example problems are:

- Bad local minima are suboptimal solution that tricks optimization in finding a better solution. following the steepest gradient might be the reason this problem, to mitigate this problem using mini batches helps, since optimizer is not following the exact steepest gradient.
- Saddle point is the opposite of local minima, but also having zero gradients can hurt the optimization process, as SGD cannot easily determine which direction to take. [18]
- When bad learning rate is set, it might result in overshooting the local minima.
- When dataset is badly scaled it can result in a narrow valley in the space of loss function. instead of quick descent downhill, the optimization descends to the valley and then goes very slowly through the valley towards minima. [19]

High-level overview of various SGD based optimizers:

- Momentum [20], it adds a portion of the previous update to current update. The momentum behavior can be described as a ball rolling from a hill and acquiring momentum slowly.
- Adagrad [27] it adapts, individual learning rate for each parameter separately by accumulating a sum of squares of all previous gradients of that parameter. This can lead to
- Adadelta [28] it improves on Adagrad by reducing its monotonically decreasing learning rate by using the only window of gradient accumulation.

2.3.4 Using hardware with high parallelization capabilities

Backpropagation is a very computationally intensive task, however by using parallel algorithms, the training process can improve a lot. Moreover, using special hardware for parallel computing such as graphics cards enables deep learning on big datasets for models involving complex tasks such as speech recognition and image classification. The recent development in parallel computational frameworks such as TensorFlow [21], which can leverage the computational power of GPU's have led to mainstream usage of neural networks.

2.3.5 Making learning scalable through batching

Learning process of deep networks is a very resource intensive task. Graphics cards involved in deep learning have limited memory. During the learning process the network containing millions of parameters needs to be stored on GPU, which leaves not a lot of space for training examples. This problem is addressed by shuffling the dataset and picking a smaller number of examples called batch which can fit in the memory of GPU. The set of batches from the shuffled dataset is called epoch.

2.3.6 Activation functions

To learn complex non-convex functions, non-linearity is introduced by using activation function. the activation function used in this work is ReLu [22]. The ReLu activation function returns positive part of its argument, leaving all negative inputs to be zero.

2.3.7 Over-fitting

The objective of training a classifier is to teach a general concept. If the task is to recognize a dog or a cat, the desired behavior of training process is learning essence of looking like a dog or a cat. Nevertheless, the neural network will try to achieve the objective of minimizing the training loss at all cost. This is usually achieved by memorizing the training examples instead of learning a general concept by the network. This is known as overfitting [23]. in ideal case, the dataset should be big enough that the network is not able to memorize training examples and is forced to generalize towards the learned concept.

2.3.8 Dropout regularization

One of the several ways to avoid overfitting is the usage of ensemble of many models. This is very computationally expensive. The best practice approach is to use dropout regularization [23]. The dropout when implemented will randomly stop a portion of activations from propagating. The intuition behind this idea is that training such a network is like training multiple network at once.

2.3.9 Loss functions

Usage of loss function depends on the task solved and it have a direct effect on the speed of convergence of training. Most basic problems are binary classification, multi-class classification and regression with their respective loss functions.

In equation 3.3 binary cross-entropy is described, where *n* is number of examples, *yi* is target label, f*(xi,θ)* is classification function with input *xi* and parameters θ.

$$-\sum_{i=1}^{n} yilogf(xi,\theta) + (1-yi)\log\big(1-f(xi,\theta)\big) \qquad (3.3)$$

For multi-class classification cross categorical entropy is widely used, which is shown in equation 3.4, where *n* is the number of examples, *yi* is target label, *f (xi,θ)* is classification function with input *xi* and parameters θ.

$$-\sum_{i=1}^{n} yilogf(xi,\theta) \qquad (3.4)$$

Square loss widely used in regression problems is in equation 3.5 where n is number of examples, y is target value and is prediction.

$$-\sum_{i=1}^{n}(y-\hat{y})^2 \qquad (3.5)$$

2.3.10 Convolutional neural networks

2.3.10.1 Convolutional layers

Convolutional layers, as the name suggests employs convolution operation and they preserve the spatial structure, this being the main difference between traditional fully connected layers of neural network. For example, an image of dimensions 32x32x3, instead of converting into one-dimensional vector of 3072 items, the image is held in original 2d structure. Applying convolutional filter, the input is transformed into a different tensor called activation map which also preserves structural properties. Since without loss of structural information, activation maps can be convolved again stacked convolutional layers can be used to reduce the dimensionality of spatial data to a low-dimensional feature rich vector space where conventional fully connected networks can be applied.

Filters are small matrices of numbers, which are multiplied by regions of input. for each pixel in the input layer, the filter is aligned by its center and then the filter is multiplied with region of input with same size as filter. this is a repeated process done for all pixels apart from those with not adequately big neighborhood, which results in an activation map of slightly smaller size. This repeated process of applying filter over the image can be

viewed as a filter sliding over the image, hence convolution. The filter will have the original depth of input, if an image is RGB, the filter will have depth of 3.
Apart from applying different filter size, the way filter is sliding over an input can be adjusted. Instead of going over all the pixels, it can be applied after every other pixel. This parameter is known as stride [17]. With stride value 1 the filter will slide over every pixel, while with stride 2, it will slide over every other pixel. Higher the stride lower will be the activation map. It's worth noting, that not every filter size, input size and stride are compatible. To solve this shortcoming and to apply filter to borders of the image, zero padding the input can be used to achieve compatible parameters, because it allows us to

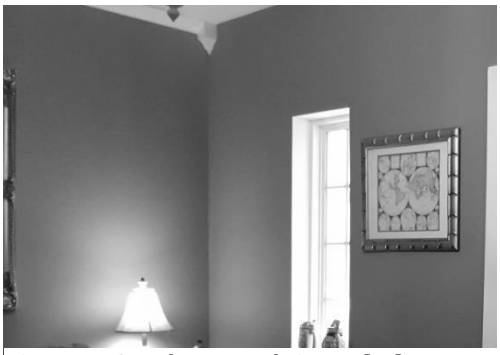

Figure 2-7 Sample gray scale image [32]

105	102	100	97	96
103	99	103	101	102
101	98	104	102	100
99	101	106	104	99
104	104	104	100	98

Figure 2-6 sample image pixel intensity values [32]

control the size of the activation map. Since higher rate of area shrinking leads to too rapid loss of information, this helps to reduce the rate of area shrinking.

In the above figure 2.7 a gray scale image, then in the figure 2.6 the pixel values of the same image can be seen. In the following figures 3.3 and 3.4, we can see the filter sliding

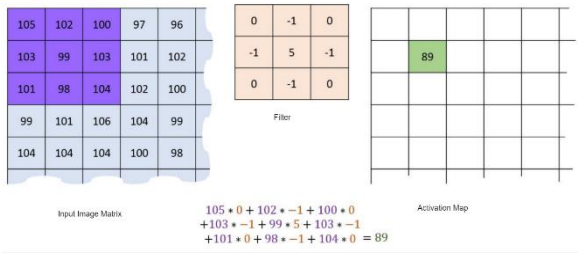

Figure 2-9 filter activation map 1 [32]

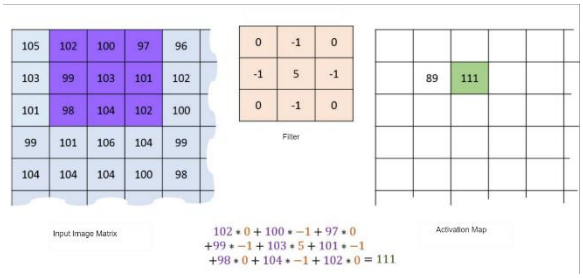

Figure 2-8 filter activation map 2 [32]

over each pixel, the resulting value can be seen in activation map. Figure 3.5 shows zero padding the input to compute the border pixels, another effect of zero padding will be the filter will slide over more input pixel resulting in the desired output of activation map.

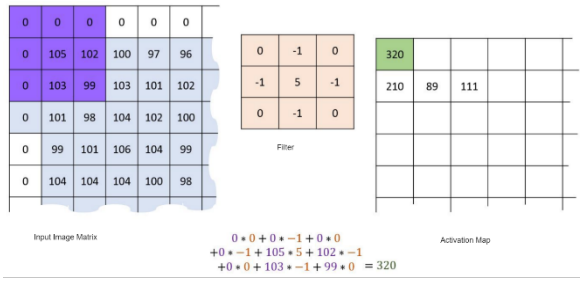

Figure 2-10 zero padding and border pixel [32]

2.3.10.2 Pooling Layers

Pooling layers can be used as an alternative approach to shrink input volume area [17]. Pooling layer carries out aggregations over regions, instead of multiplying the trained weights with filters. Usually the aggregations done is the maximum of the region. The effectiveness of max-pooling layer in classification task can be described as, it's not of concern where feature in the maps is found, but as long as the feature is found. By taking the maximum of a region of activations neglects unimportant parts of the region and reports presence of the feature in all region. If instead of maximum averaging is used, the fact that feature was not detected in the rest of the region would have a negative effect on significant activations. Pooling layer does not have any trainable parameters. mostly pooling layer is used to down-sampling, for this reason stride is set to avoid overlapping regions.

An input is composed of different features and a single filter looks for a feature in the input, so many filters are used to find many features in an image. With this each filter results in its own activation map, together resulting in a stack of activation maps called output volume. Intuitively, as the input is being transformed from input image to features across the network, the area of input volumes decreases with applied stride or pooling, and the depth of input volumes can both increase and decrease based on the number of filters used in convolutions. In short, single convolutional layer requires four hyper parameters, number of filters, filter size, stride and amount of zero padding.

2.3.11 Deep Convolutional neural network architectures

Deep convolutional neural networks are composed of layers that filter or convolves the inputs to get useful information. These layers have learnable parameters, these filters are automatically adjusted to extract the useful information about a given task without feature selection. Deep CNN is proven to work better with images. Simple neural networks don't work well with image classification problems.

We will describe some of the important deep CNN, Alexnet, Googlenet and Vgg. Each of which has won the ILSVRC competition of ImageNet.

2.3.11.1 Alexnet

Alexnet [16] was the first large-scale CNN that was able to win ILSVRC [24] competition in 2012 by surpassing other methods by a significant margin. This lead to a new wave of research into CNN, which is the basis for so many improvements in deep learning and is still going today. Alexnet was the first network to use ReLu activation functions, since the training was done on GTX 580 GPU's with 3 GB memory, two GPU's were used. Alexnet contains 5 convolutional layers and 3 fully connected layers. ReLu is applied after convolutional and fully connected layer, dropout is applied before the first and the second fully connected layer [16].

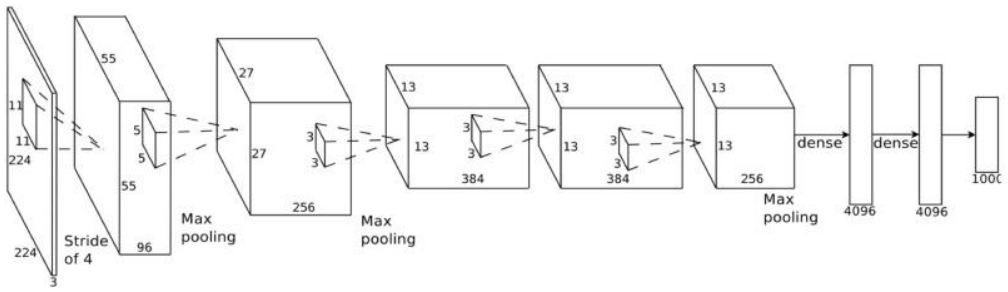

Figure 2-11 AlexNet Architecure schema taken from [31]

2.3.11.2 GoogLeNet

The idea that deeper is better was confirmed by two architectures in the ILSVRC challenge in 2014. GoogleNet [25] by Google and VGG [26] from oxford. GoogleNet is also a deeper network with 22 layers. However, it uses inception module to reduce computation

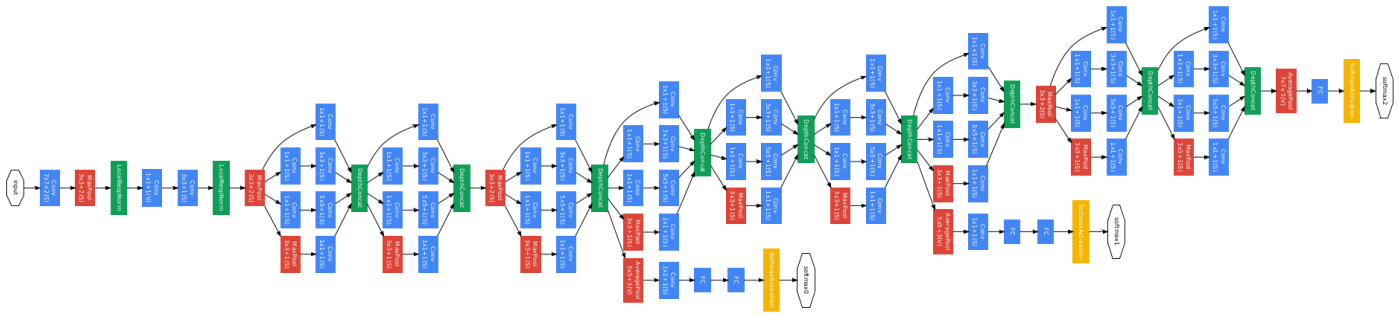

Figure 2-12 Goolgenet Architecture. [33]

difficulty. Being deeper than Alexnet, GoogleNet still has twelve times fewer parameters than Alexnet. The inception module is local network topology, it can be seen as a network within a network as can be noticed in the figure 2.12 and figure 2.14 shows the inception module. The input of these modules is fed into multiple different layers such as convolutions of different sizes and pooling. To get the resulting activation volumes of same size, stride and padding is setup. The activation volumes are concatenated depth-wise, resulting in the output of the module.

Since pooling is one of the layers inside the module, and the output is concatenated with other layers, this will result in the depth of output always being higher than the input. Also, since inception modules are stacked, this would lead to rapid depth increase and high computational difficulty. To overcome these problems using bottleneck layers to reduce dimensionality of input or output layers within inception module. Bottleneck layer implemented by 1x1 convolution layer with number filters equals to desired depth. the input first runs through a small conventional network of convolutions and pooling before entering the stack of inception modules. There are also two auxiliary outputs branches to inject gradients in earlier levels, which are discarded during inference.

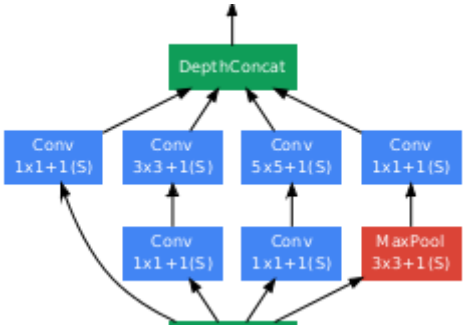

Figure 2-14 Goolgenet Inception module. [33]

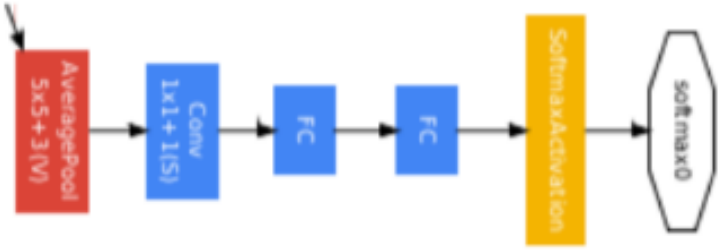

Figure 2-13 Goolgenet Auxilary output. [33]

2.3.11.3VGGNET

VGGNET [26] was the runner-up at the ILSVRC 2014 competition, it has 16 convolutional layers and because of its uniform architecture its very appealing. Similarly, to Alexnet, it has lots of filters but only 3x3 convolutions. It was trained on 4 GPUS for 2-3 weeks. it's a preferred choice for feature extraction, its weight configurator is available to the public and used in many applications and challenges as a baseline feature extractor. But it has a very large number of parameters, 138 million which can be a bit challenging to handle.

Figure 2-15 VGGNET 16. [34]

2.3.12 Tensorflow

TensorFlow is a powerful library developed by Google and released in November 2015 to carry out large-scale numerical computation. deep neural networks can be implemented and trained with ease. TensorFlow provides the capability to define functions on tensors and tensorflow will compute their derivatives. We will make use of TensorFlow in this thesis to achieve the goal of training deep neural networks.

The computations in Tensorflow are executed in a graph, Computations are described in the graph, A node represents mathematical operations, while the edges represent the multidimensional data arrays (tensors) and forms the communication between the nodes. The graph can then be executed on multiple devices.

```
def train(args):
        with tf.Graph().as_default(), tf.device('/cpu:0'):
        images, labels = data_loader.read_inputs(True, args)
        #graph creation
        logits = arch.get_model(images, wd, is_training, args)
        .
        .
        .
        for epoch in xrange(start_epoch, start_epoch + args.num_epochs):
            #actual training
            _, loss_value, top1_accuracy, topn_accuracy = sess.run(
                    [train_op, cross_entropy_mean, top1acc, topnacc], options=run_options, run_metadata=run_metadata)
```

Figure 2-16 Graph creation and actual training process

Researchers and engineers from the Google Brain Team within Google's Machine Intelligence research organization for the purposes of conducting machine learning and deep neural networks research, but because of the system being general enough it can be applied in variety of other domains also.

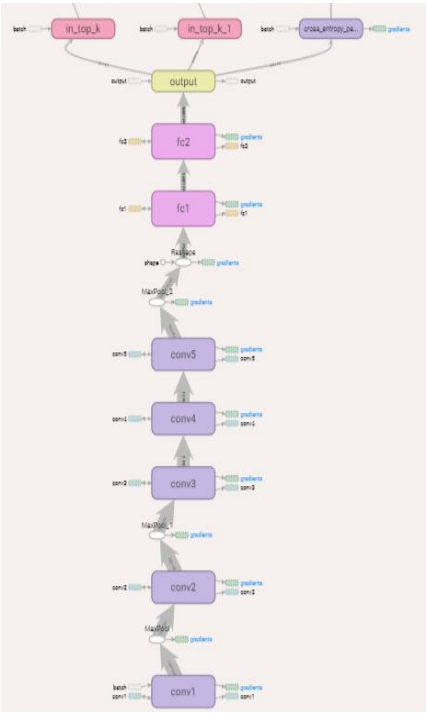

Figure 2-17Alexnet graph

Tensorflow has some advantages over other machine learning frameworks, because of better performance in complex tasks [6], and better compilation time [7].
Besides being suitable for deep learning, Tensorflow has support of GPU processing and has shown good performance while solving complex tasks.
Few of the important points of Tensorflow is described below.

- Multi GPU Support
- Training Across Distributed resources, (i.e., cloud)
- Queues for data loading pre-processing.
- Visualizing the graph using tensor board, when building, debugging new models.
- Model Checkpointing.

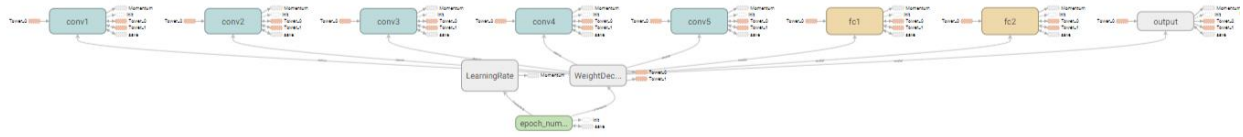

Figure 2-18 Tensorboard model graph visualization

2.3.12.1 Advantages of Tensorflow

- **Flexibility:** The computations are needed to be expressed as a data flow graph to use Tensorflow. Tensorflow is a highly flexible system, providing multiple models or multiple versions of the same model, and can be served simultaneously. Because of the modular architecture of Tensorflow, some parts can be used individually or can be used all together. Such flexibility enables non-automatic migration to new models/versions, and A/B types of testing with experimental models. It also allows deployment of computation to one or more CPUs or GPUs in desktops, server or mobile device with a single API.
- **Portability:** Tensorflow is very portable, in the sense that we don't need dedicated hardware to run/test models, it runs on GPUs, CPUs, desktops, servers, and handheld devices. A trained model can be deployed on mobile devices, serving the true purpose of portability.
- **Research and Production:** Tensorflow can be used to train and serve models in real time, meaning no rewriting of the code, and industrial researchers can implement their ideas quicker. Also, academic researchers can share codes directly with greater reproducibility. This way it speeds up the process of research and production.
- **Auto Differentiation:** Tensorflow has auto differentiation capabilities from which gradient-based machine learning algorithms can take benefit. In Tensorflow we can define computational architecture of the predictive model, combining with objective function and adding data, Tensorflow manages derivatives computing processes automatically. We can compute the derivatives of some values with respect to some other values results in graph extension and you can see exactly what's happening.
- **Performance:** Tensorflow allows to make the most of available resources/hardware, because of its advanced support for threads, asynchronous computation, and queues.

2.3.13 Other Frameworks

Other frameworks that compete with Tensorflow are lasagne, Theano, Blocks.

Theano [38] is python library which allows to define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays efficiently. It works with GPUS. It was developed by University of Montreal's lab Mila. The API is quite low level. Theano trades ease of use with flexibility.

Lasagne [39] is a light weight library for building and training neural networks in Theano. it offers abstractions of top of Theano to make it more suitable for deep learning. It allows users to work with layers, providing building blocks like "Conv2DLayer" and "DropoutLayer". Lasagne provides a lot of common components to help with layer definition, layer initialization, Model monitoring and model training.

Blocks [40] like lasagna adds a layer of abstraction on top of Theano to facilitate simpler and cleaner definitions of deep learning models as opposed to writing raw Theano. it is a bit more flexible than lasagne, but with a bit more learning curve to use effectively.
Keras [41] is a high level neural networks API, it can run on top of Tensorflow, or Theano. it was developed keeping fast experimentation in mind. It allows for fast and easy prototyping through user friendliness, modularity and extensibility. It runs on CPU and GPU.

2.3.14 Deep Learning and GPU's

2.3.14.1 Introduction

GPGPU a parallel programming configuration based on GPUs and CPUs to process and analyze data the way image or other graphics are processed. GPGPUs were created for general graphic processing initially, but later found as a useful fit for scientific computing as well. The reason behind, most of graphic processing involves large matrix operations and computation.
Use of GPGPUs for scientific computing started in 2001 with the implementation of Matrix multiplication. One of the first algorithms to achieve better performance was LU factorization in 2005. But at the time, knowledge of how level graphic processing and coding for GPUs was needed. That will change with the high-level language CUDA, released by Nvidia in 2006. CUDA helped researchers with writing program for graphic processors in high-level language, this significantly change how researchers interacted with GPUs. [10]

2.3.14.2 GPU's vs CPU

CPUs are designed to handle general computing tasks, GPUs in comparison are less flexible, however GPUs by design can compute the same instructions parallel.
GPUs have additional advantages in comparison to CPUs, including more computational units and a higher bandwidth for retrieving from memory. Going further, graphics intensive applications (i.e. Convolution Neural Networks) can take advantage of GPU graphics specific capabilities to further speed up the computation process.
Both CPU and GPU are capable of handling graphical operations, GPU speeds up or accelerates the graphical computations very well, because of distributed/parallel nature of the architecture. [13]
In comparison GPUs have almost 200 times more processors per chip than a CPU. For example, an Intel Xeon Platinum 8180 Processor has 28 cores, while an NVIDIA Tesla K80 has 4,992 CUDA cores. Although a CPU core is more compared to GPU core, in machine learning applications most of this power goes unused. CPU core by design is built for broad variety of tasks (e.g., render a webpage, drive word processors and enterprise

software, manage peripherals) in addition to performing computations, while on the other hand GPU core is optimized exclusively for data computations. Because of this singular focus, a GPU core is simpler and needs a smaller die area than a CPU, allowing the placement of many more GPU cores on a single chip.
In Machine Learning applications, in which large numbers of computation on large amount of data is required, can gain (i.e. 5 to 10 times) performance improvements when running on GPU versus a CPU. [11]

2.3.14.3 GPU Computing and Deep Learning

GPUs has turned out to work a lot better in Deep learning, the reason behind is a, CPUs can fetch small packages/blocks of memory faster whereas GPUs have a high latency, making it slower at this type of task. But GPUs are ideal when it comes to fetching/loading large amounts of memory and the best GPUs can fetch larger bandwidth (e.g. Nvidia Titan V has a bandwidth of 900GB/s) [15]. which is way higher than the best CPU with 50GB/s memory bandwidth.
To resolve the latency issues, more than one processing unit is used. A GPU core is composed of thousands of cores to solve a task involving large amounts of memory and matrices, only wait for initial fetch is required. Each subsequent fetch will be significantly faster because the unloading process is time consuming that in turn all the GPU must queue to continue the unloading process. With so much processing power, the latency is effectively masked allowing the GPU to work with high bandwidth. this is called thread parallelism, and this is another reason why GPUs have an edge over CPUs in deep learning. [12] Deep Neural Networks are structured in a very uniform manner, such that at each layer of the network, thousands of similar artificial neurons perform the same computation. Because of this the structure of a DNN fits very well with the computation in which GPU is efficient and faster. Nvidia released a new GPU with a new architecture called Volta specifically built for Machine Learning/Deep Learning, there are 5120 cuda cores as well as 640 Tensor Cores and delivering 110 deep learning Teraflops which will significantly speed up the Machine Learning process. [14]

Chapter 3

3 Implementation

This chapter describes the approach used to solve the problem of the unknown execution time of training an ImageNet classifier and the per-layer time of the training a CNN model in a cloud environment. Machine learning was once considered out of reach of the masses because of budgetary restrictions. Cloud providers ability to provide machine learning services makes this technology within the grasp of students, researchers and even small enterprises. Cloud supports massive data storage capacity, scalable compute power, and embedded graphics processing power to handle data stores and ML algorithms, and with GPU acceleration, neural network training is 10-20 times faster than CPUs. This capability of cloud and commercialization of cloud have brought forward this advantage, that dedicated hardware like huge memory or GPUs which if bought separately and then building the infrastructure, needs a large investment. What cloud has done is that it provides pay per use model which greatly removes these worries of not having dedicated hardware or memory.

We will propose an approach to solve the problem of unknown execution time of the layers of CNN, this approach will be implemented as part of atmosphere [37] project framework. Atmosphere [37] is a joint project between EU and brazil which aims to provide a solution to assess trustworthiness of cloud applications dealing with data and supports the development of more trustworthy cloud applications. The implementation will focus on integrating this application into atmosphere framework. the per-layer information can be used to build machine learning models to regress application training time.

This chapter starts with Section 3.1, which explains the problem statement and Section 3.2 explains the proposed approach. and Section 3.3 explains the uses cases and the following Sections 3.4, 3.5 details the implementation.

3.1 Problem Statement

ImageNet [24] is a database used for training and testing of neural networks for visual recognition problems. The training data, the subset of ImageNet containing 1000 categories with 1.2 million images, and with a validation data containing 50,000 labeled

images. Training on such a large dataset requires large memory and time. This can be achieved by using dedicated hardware and a framework which allows the usage of parallel computing over multiple GPUs.

The aim of this thesis is to build an infrastructure to collect information about different layers of a CNN during training, which will be used to develop performance models for the prediction of training time based on the structure of the model. It involves developing an application for training ImageNet with different models with multiple GPU support and that can compute the time it takes for each layer during the training process of a CNN model. This will give us an insight into the layer-wise time distribution of the training and more specifically how much time is spent during the forward pass and backpropagation. This thesis aims to achieve the following.

- Building a multi-gpu ImageNet training application.
- Computation of per-layer data of the training model.
- Computation of time spent in forward pass and back propagation.

3.2 Proposed Solution

There are many deep learning frameworks which can achieve the task of training an image classifier. we will use tensorflow to achieve the goals described above. Because of TensorFlow's strong support for deep learning, ease of use and because tensorflow enables us to profile the time of individual operation during training a model. TensorFlow also allows the flexible creation of deep learning architectures, by using basic building blocks. In our proposed solution we used tensorflow to implement a training framework, which can train different architectures on ImageNet dataset. The application makes use of tensorflow capability to train over multiple GPUs, it runs training operation on multiple GPUs in parallel. More over Tensorflow has the built-in functionality to extract the memory and time profiling information of the training process. we will use this functionality to profile training at each iteration.

The profile output file holds the profiling information for each operation not properly grouped. Tensorflow provides the ability to group operations with variable scoping [29] as can be seen in figure 3.1, by using variable_scope to group each layer of the CNN model which helps distinguish each operation in the profiling output file, which uses paths for the different nodes of the graph to organize the profiling information. By using the names

```
with tf.variable_scope('fc1'):
  network = tf.nn.dropout(network, dropout_rate)
  network = fullyConnected(network, 4096, wd= wd)
  network = batchNormalization(network, is_training= is_training)
  network = tf.nn.relu(network)
with tf.variable_scope('fc2'):
  network = tf.nn.dropout(network, dropout_rate)
  network = fullyConnected(network, 4096, wd= wd)
  network = batchNormalization(network, is_training= is_training)
  network = tf.nn.relu(network)
```

Figure 3-1 Tensorflow Variable Scoping

from variable scoping each layer can be distinguished in the profile output file, and we can collect the computation time from these profile output files for each layer. the collected time of each layer consist of the time spent by each layer in the forward pass and backpropagation the time to update momentum is also stored.
The application developed will be integrated into the atmosphere framework. which can help launch the application with different configuration settings. It is worth noting that the atmosphere framework's pytorch imagenet training application was available to us. help is taken from pytorch implementation for some functionality of this application, e.g synchronizing the files with remote location, and computation of parameters.
The following sections will describe the implemented solution in more detail.

3.3 Use Cases

This section describes the use case scenarios for the implemented training framework in TensorFlow. This gives an idea to the reader as to what this program does.
There are two primary scenarios explained in this section, **run training with profiling**, **run Training without profiling,** and two secondary use cases **Collect Profiling information** and **Collect experiment data.**
These scenarios are explained in the following tables. In both primary use cases, the user is expected to provide a configuration file containing the experiment configuration i.e. location of data set, remote location of data set, architecture type, batch size, ImageNet classes to use, logging flag etc. Also, in the secondary use cases if not used within the training process, the user is expected to provide profiling output directory, training model and the execution output along with the GPU in use of the training needs to be passed to the second secondary use case.

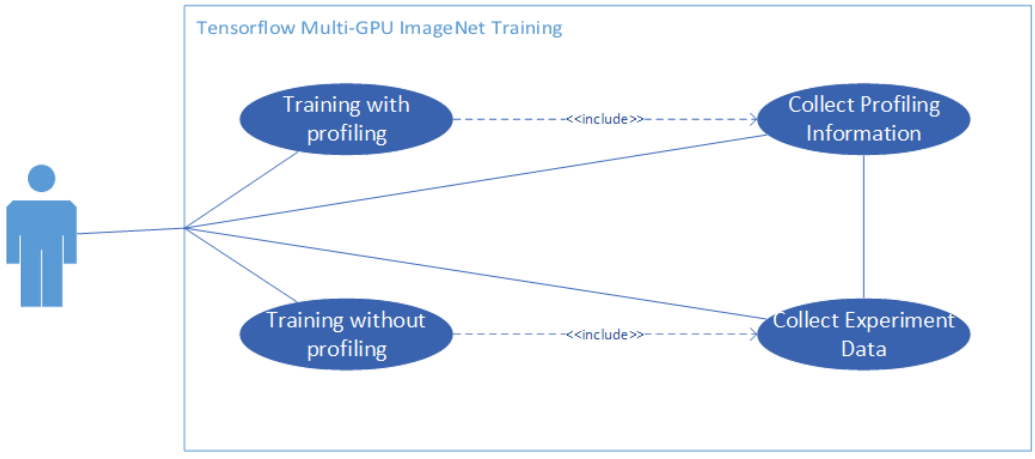

Figure 3-2 Use Cases

Use Case Name	Run Tensorflow training with profiling.
Description:	This will run the training process with profiling enabled, saving the profiling output of each iteration.
Primary Actor	User
Precondition	The user is expected to provide a configuration file.
Flow of Events	1- User provides a configuration file. 2- The script parses the file and retrieves/synchronizes the experiment data from server and creates label dictionary of the classes for training. 3- A batch of pre-processed images are passed to the training step 4- Profiling information is saved after each training step. 5- Collection of layer data information takes place after training; the profiling files are parsed, and the time information is extracted from it and saved into a csv file. 6- Collection of experiment data is collected and saved into a csv file.

Table 3-1 run Training with profiling Use Case

Use Case Name	Run Tensorflow training without profiling.
Description:	This will run the training process without profiling. Not computing and storing the per-layer information.
Primary Actor	User
Precondition	The user is expected to provide a configuration file.
Flow of Events	1- User provides a configuration file. 2- The script parses the file and retrieves/synchronizes the experiment data from server and creates label dictionary of the classes for training. 3- A batch of pre-processed images are passed to the training step 4- Collection of experiment data is collected and saved into a csv file.

Table 3-2 Training without profiling Use Case

Use Case Name	Collect Profiling information
Description:	User can also use collection of profiling information without the training, by providing the profiling output directory.
Primary Actor	User
Precondition	Profiling output files must be provided to be used stand alone.
Flow of Events	1- User provides the profiling output files. 2- Layer time for each iteration is computed from the files 3- Save the extracted information in a csv file.

Table 3-3 Collect Profiling data Use Case

Use Case Name	Collect Experiment information
Description:	We can collect the experiment information after doing the training and providing only the output of the training script to this method.
Primary Actor	User
Precondition	Experiment output must be provided
Flow of Events	1- User provides the training script output file. 2- Per iteration training time is collected and stored in another csv. 3- Save the configuration information along with the training time, missing time at 25^{th}, 50^{th} ,75^{th} and 100^{th} percent iteration in a different csv.

Table 3-4 Collect Experiment information use case

3.4 Class Diagram

tensorflow the main class of the application which holds all the different functionalities of the implemented solution to the problem described above. This class takes care of starting the training process and if enabled collects the profiling information and experiment information. This class contains a few functions. The ComputeParameters function computes the parameters passed to the tensorflow class, and it loads the XML configuration file and returns it as a dictionary. The configuration information is extracted from the XML configuration. Another function synchronizeFiles which takes input the remote server information from the configuration file and synchronizes the files with the remote server.

The dataHelper class contains functions that help prepare the input files for loading, tensorflow class uses the function buildTraining and buildValidation of dataHelper class to create the input file for the images which maps the path of the image with its corresponding label of the class of the image. tensorflow class uses the extracted information from the configuration file and passes it as parameters to the training class tflow_train.

The training class tflow_train takes care of the training process. it first loads the data for training with the help of data_loader class by passing the input file to the read_input method of the data_loader class, the data_loader class performs pre-processing on the images and returns an images batch of a specified size to train the model on.

The selected architecture model class is loaded by with the help of get_model method of the arch class and is called by the training and the validation script to load the model. The model classes are built using different functions from the class named helperClass.

Another class tflow_eval which is used by tensorflow class to start the validation after the training completes which performs the validation on the validation set after the training is completed.

tensorflow class also contains the functions ComputeLayerData and CollectData, ComputeLayerData function collects the data from the profiling output files, it takes the profiling output directory and model name as arguments. The function CollectData is used to collect the information like preparing an output file for the storage of iteration times of the training and validation process and storing the configuration information along with overall training time, validation time at 25%,50%,75 and 100% and the information about output paths are stored by this function.

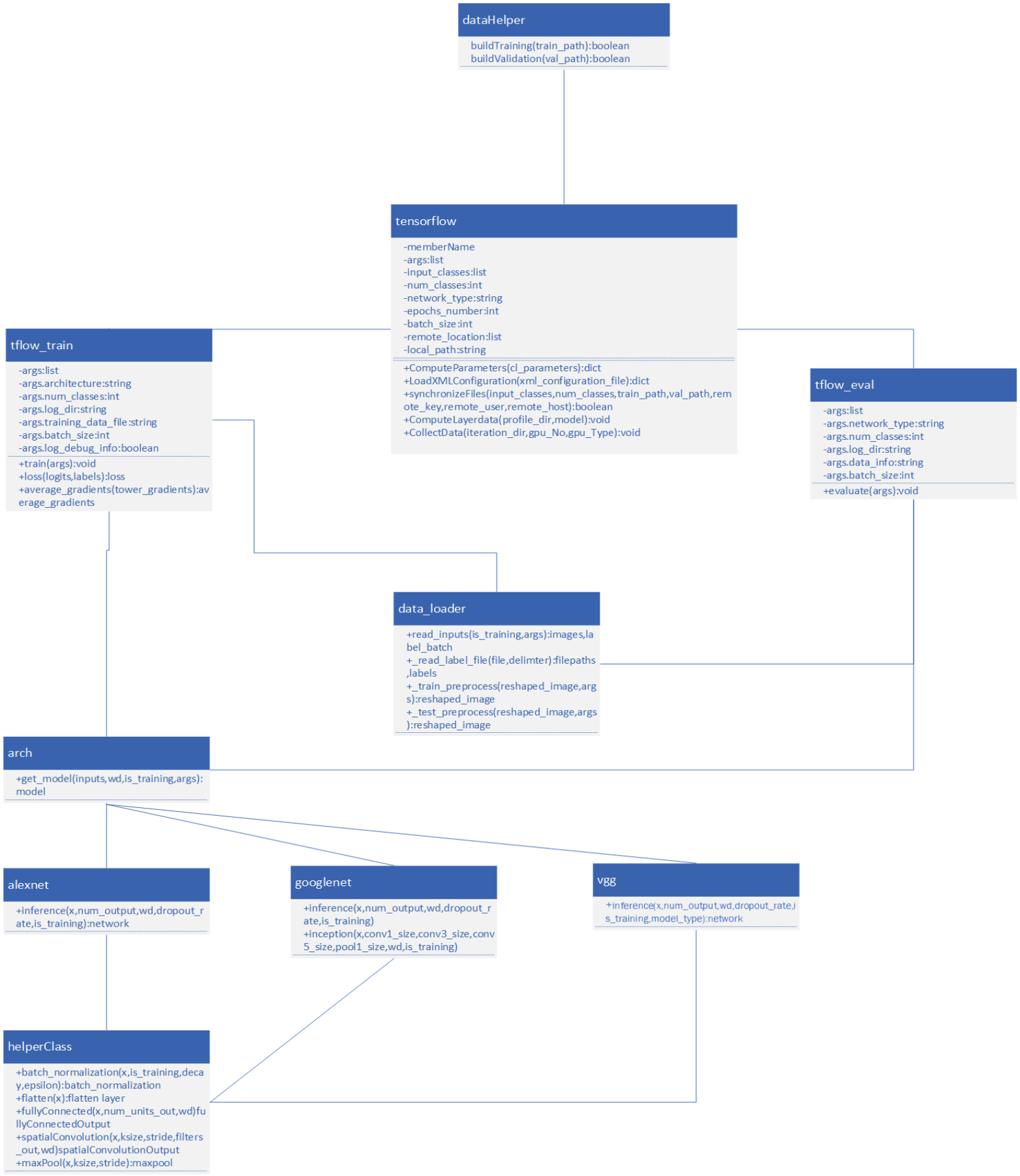

Figure 3-3 Class Diagram

3.5 Sequence Diagrams

A sequence diagram is used to demonstrate the interactions between different components of the tensorflow training application. The sequence diagram is equivalent to the Primary Use case.

The experiment is launched by a user, the launch_experiment a class of atmosphere framework which creates a virtual machine on Microsoft Azure and starts the launch_local_experiment on the virtual machine. The launch_local_experiment class takes the name of the application and the parameter list, based on which it launches an application on a cluster. The application is executed for several times based on the repetition number received from the user. For each repetition a separate directory is created by the launch_local_experiment, launch_local_experiment launches the tensorflow application with the configuration parameter and output directory.In the sequence diagram the sequence of activities for our implemented solution is shown.

When the tensorflow application starts, before starting the training the XML configuration file needs to be loaded and the information needs to be extracted from it, tensorflow uses the method computeParameters, which takes the name of the configuration file and returns the configuration as a dictionary. Once the configurations are loaded, data of the experiment needs to synchronize with the remote server. The configuration file holds the class names and the remote data location information. tensorflow calls its own method synchronizeFiles with the remote location information and class names from the configuration file. Once the files are synchronized. We need to prepare input files for the training and validation process of the data of the experiment, for this tensorflow provides the training and validation directory to dataHelper, this results in the generation of input files with the images names with its corresponding label, these files are needed to load the data during the training and validation process.

Now tensorflow is ready to start the training process, tensorflow provides the configuration of the experiment extracted from the configuration file and provides it to tflow_train. tflow_train performs the actual training and initially it loads the data required for the experiment, for this tflow_train uses the input file generated by the dataHelper class and passes the input file path to data_loader. data_loader loads the data and returns a batch object. Which generate the specified batch size of images for training iteration. When the data is loaded and ready, next the model is loaded by tflow_train passing the name of the model with the is_training flag set to true to arch class get_model method. When the model is loaded the training starts. As can be seen in the sequence diagram figure, there are two loops inside the training process, one for the number of epochs, and one for the number of iterations. When profiling is enabled, during each iteration of the training the profiler information is stored in the directory of the application repetition, which will be used to collect per layer data. At each iteration of the epoch, iteration number, loss, top1 accuracy

and topn accuracy is printed. when the training loop runs for the computed number of iterations, the trained model is saved.

After training completes we need to run validation with the validation set, tensorflow calls tflow_eval with the validation input file and the checkpoint path, and with the other necessary arguments for validation such as batch_size, architecture name, this starts the validation process. For the validation to start, data and model needs to be loaded, tflow_eval loads the data by providing data_loader the input file, which pre-processes and loads the data for validation. The selected model is also loaded by tflow_eval calling arch class get_model method with the is_training flag set to false. The validation process will run for several iterations depending on the number of images and batch size.

Once the validation process is completed. tensorflow provides the profiling output directory and the training model name to its own method ComputeLayerData, which parses each iteration profile output file and collects the computation time of each layer of the selected model and stores the information in the output directory for the experiment. Similarly, tensorflow provides the std_output file path of the training to CollectData function, the information of each iteration of training is computed from the std_output file of the training script and stored in the output directory of the file.

The only difference in the sequence of activities in profiling enabled and disabled is that during the training process no profiling output will be stored. And at the end of experiment ComputeLayerData method will not be called. The rest of the sequence of activities is the same.

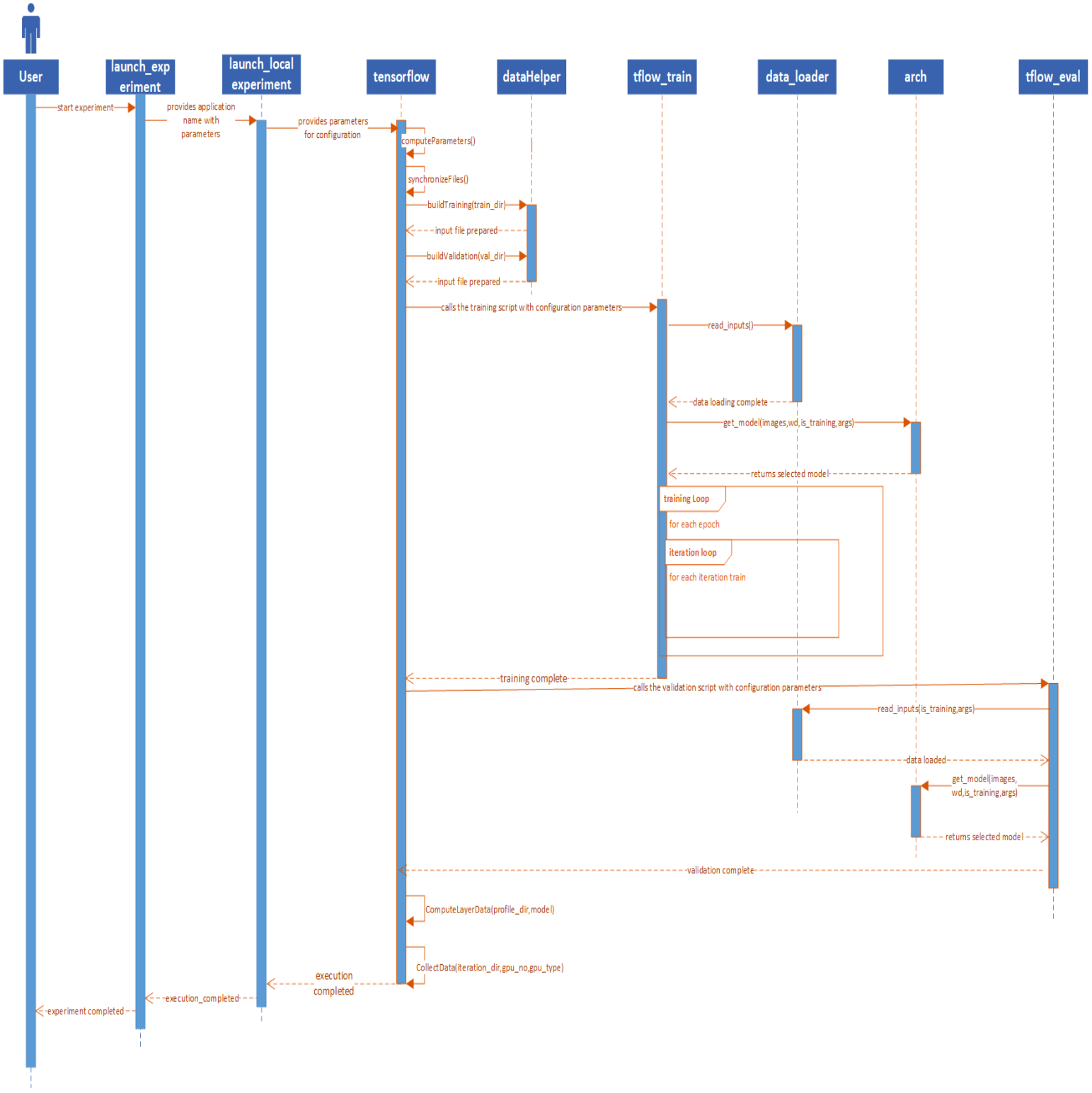

Figure 3-4 Sequence diagram

Chapter 4

4 Experimental Results

This chapter illustrates the behavior of the of Alexnet and VGG-11 trained on the ImageNet dataset.
In the following section we will describe the test conducted to verify the results. The Section 4.1 describes the machine used for testing, the following Section 4.2 describes the experiments overview, Section 4.3 reports the result of the base test while in Section 4.4 details tests related to Per-layer time in Section 4.5 the analysis of the result is presented. In the final Section 4.6 the results of this approach are discussed.

4.1 Experiments Machine Setup

Experiments were conducted on a cloud machine with two Nvidia Quadro P600 [29] with 2GB GDDR5 memory each, Intel Xeon Silver 4114 2.20 GHz [31] processor and 48 GB of ram.

4.2 Experiments Overview

Two kinds of tests were conducted, first test was to measure the time difference between the training with profiling enabled and without profiling. The second test was about the computation of per-layer time distribution of the training process.
The following section will describe the tests in detail.

4.3 Time Difference

Alexnet and VGG-11 were trained with 10 classes of ImageNet, initially with profiling enabled and later with profiling disabled. In the following figures, the tests are average time of 3 runs on Alexnet and VGG-11. As can be seen in figure 4.1 below the time difference between running a training with profiling enabled and disabled is illustrated in which we can see that profiling adds significant time to the whole training process in comparison to the run when no profiling is enabled.
The percentage increase for the test time of Alexnet from no profiling to profiling is 245 percent, while in the case of VGG-11 the increase is 235 percent.

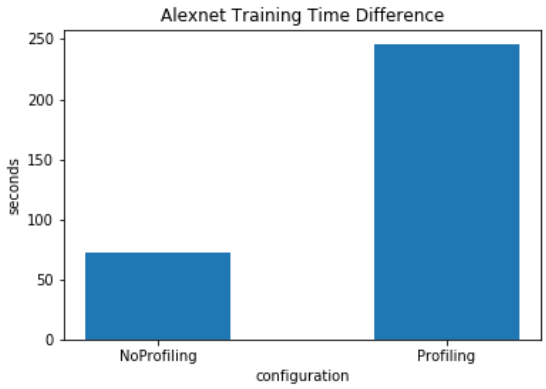

Figure 4-1 Alexnet Profiling vs No profiling time

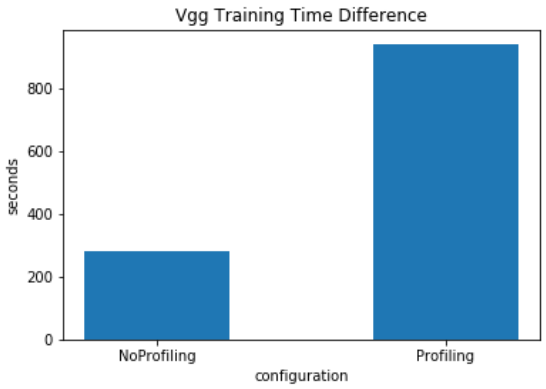

Figure 4-2 Vgg profiling vs No profiling time

4.4 Per-Layer Time Computation

For testing the per-layer time computation from the training process Alexnet was trained on 10 classes from ImageNet dataset first with 64 batch size and then with 128 batch size. How batch size affects the time spent on each layer, the time distribution of different layers, how much time is spent during each layer of the training model, followed by the time spent in forward pass and back propagation of each layer are also illustrated in the following. The comparison and percentage error between profiling and computed time of iteration is also shown in the plots.

4.4.1 Alexnet batch size 64

In the following figures from 4.3 to 4.12 we can see layers time distribution on both GPU's.

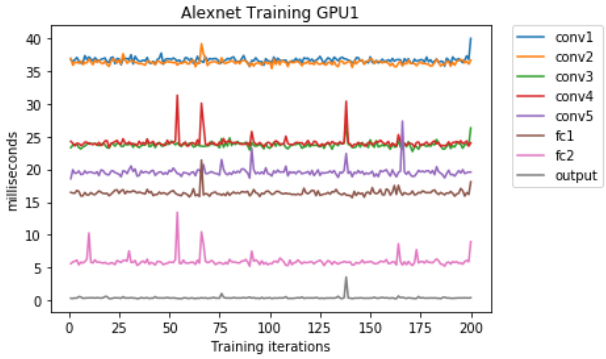

Figure 4-4 Alexnet-64 Layer-Time vs number iteration GPU1

Figure 4-3 Alexnet-64 Layer-Time vs number iteration GPU2

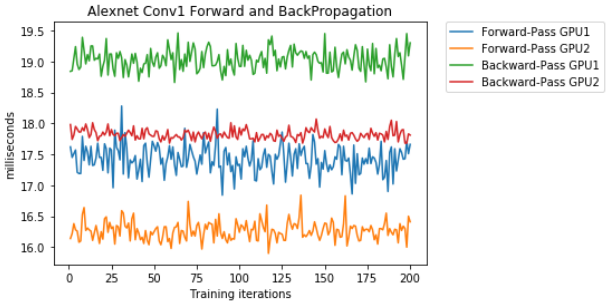

Figure 4-6 Alexnet-64 Conv1 Layer-Time vs number iteration

Figure 4-5 Alexnet-64 Conv2 Layer-Time vs number iteration

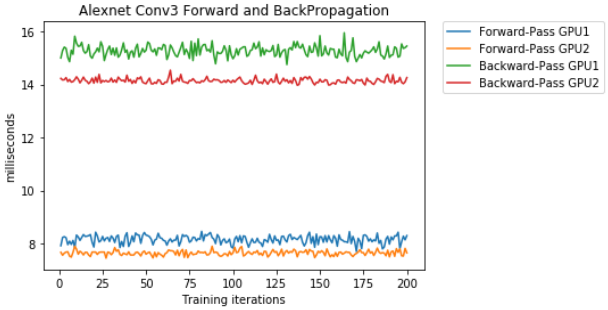

Figure 4-12 Alexnet-64 Conv3 Layer-Time vs number iteration

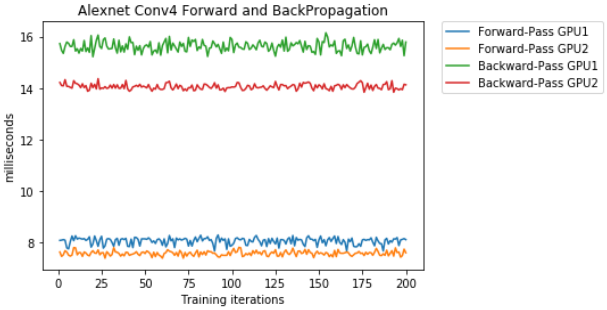

Figure 4-8 Alexnet-64 Conv4 Layer-Time vs number iteration

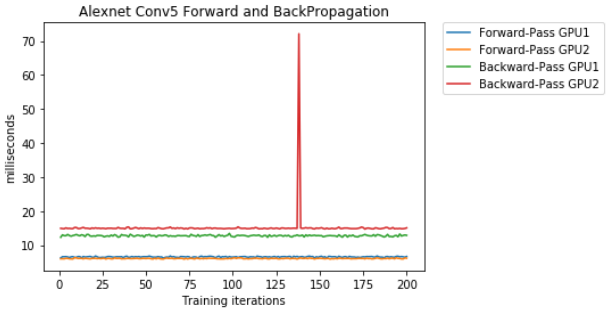

Figure 4-11 Alexnet-64 Conv5 Layer-Time vs number iteration

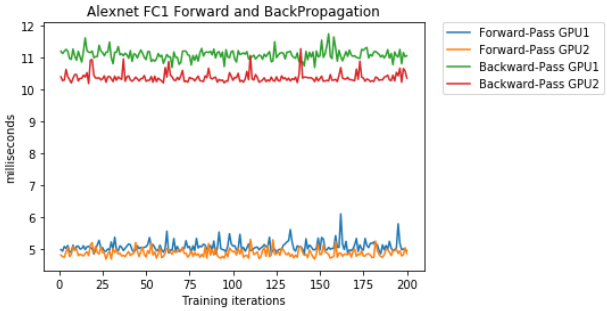

Figure 4-7 Alexnet-64 FC1 Layer-Time vs number iteration

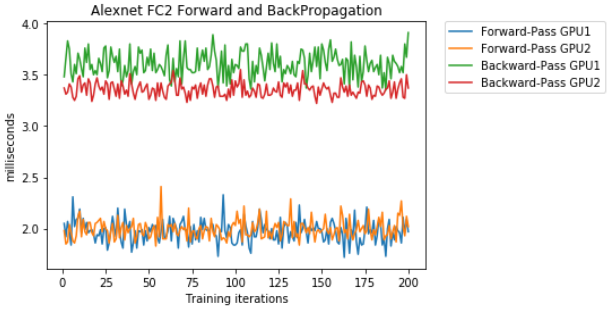

Figure 4-10 Alexnet-64 FC2 Layer-Time vs number iteration

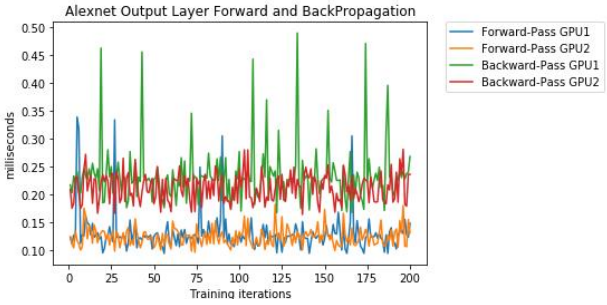

Figure 4-9 Alexnet-64 Output Layer-Time vs number iteration

The time of profiling is compared with the iteration time computed at each iteration, based on which we have calculated the percentage error which is illustrated in the following figures no 4.13 and 4.14.

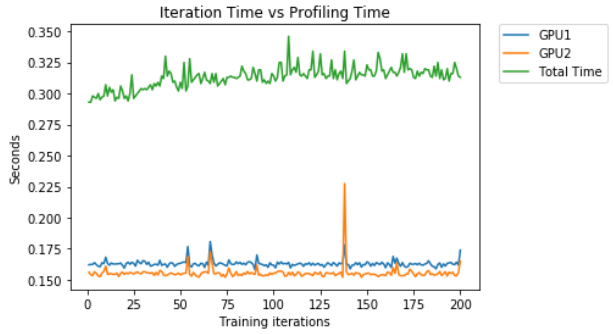

Figure 4-14 Alexnet-64 Total time vs Profiling time

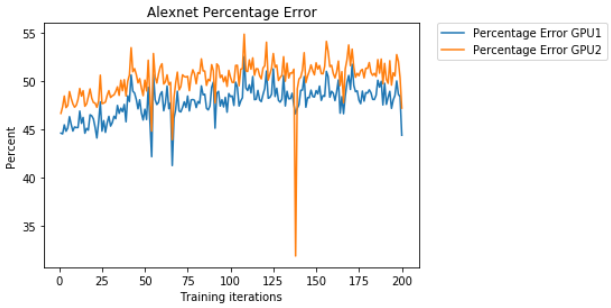

Figure 4-13 Alexnet-64 Percentage Error

4.4.2 Alexnet batch size 128

The following figures illustrate the tests conducted with batch size 128, the time distribution of layers with both GPUs can be seen.

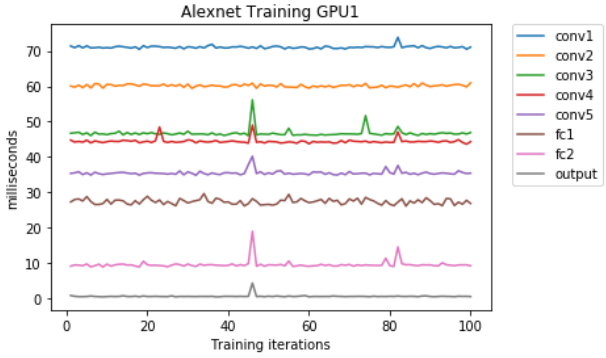

Figure 4-18 Alexnet-128 Layer-Time vs number iteration GPU1

Figure 4-16 Alexnet-128 Layer-Time vs number iteration GPU2

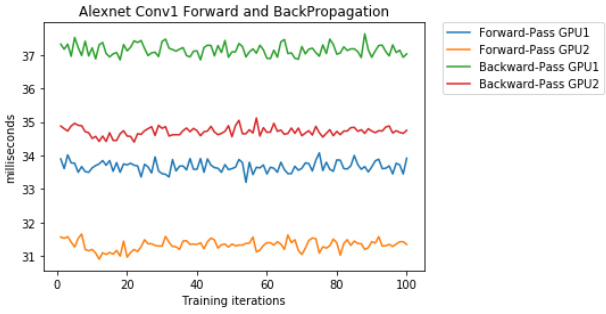

Figure 4-17 Alexnet-128 Conv1 Layer-Time vs number iteration

Figure 4-15 Alexnet-128 Conv2 Layer-Time vs number iteration

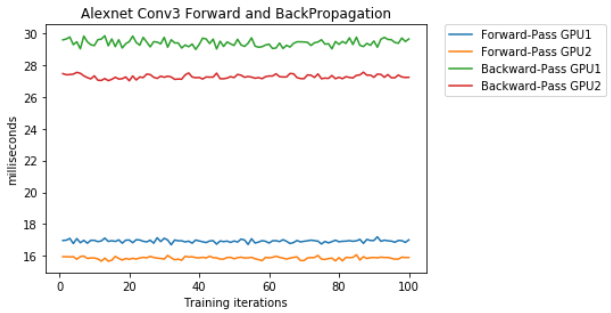

Figure 4-20 Alexnet-128 Conv3 Layer-Time vs number iteration

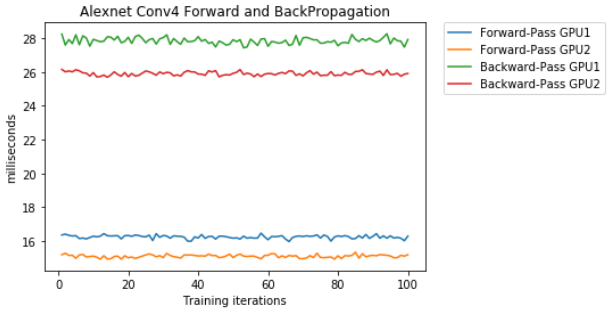

Figure 4-19 Alexnet-128 Conv4 Layer-Time vs number iteration

Figure 4-22 Alexnet-128 Conv5 Layer-Time vs number iteration

Figure 4-21 Alexnet-128 FC1 Layer-Time vs number iteration

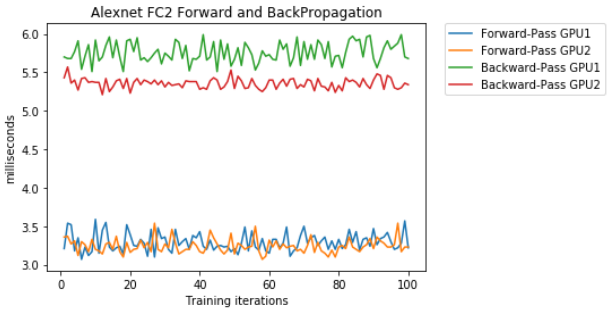

Figure 4-24 Alexnet-128 FC2 Layer-Time vs number iteration

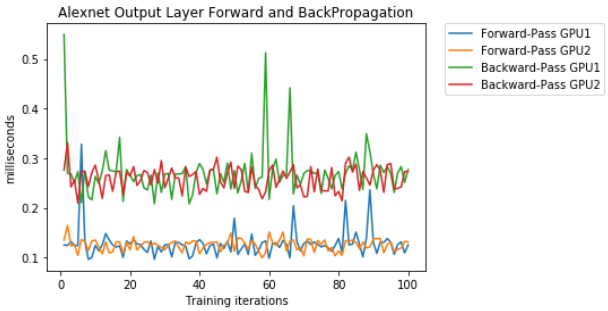

Figure 4-23 Alexnet-128 Output Layer-Time vs number iteration

Similarly, in the case of batch size 128, the profiling time is compared with the iteration training time and the percentage error is illustrated in the following figures.

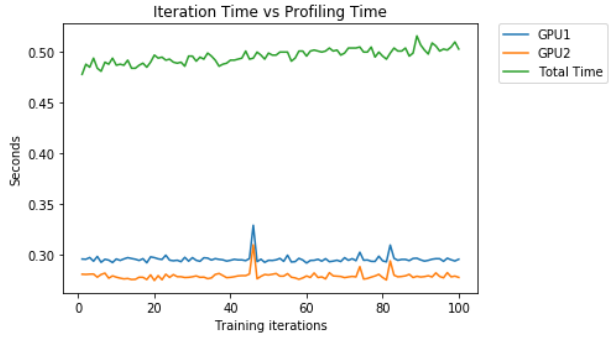

Figure 4-26 Alexnet-128 Total time vs profiling time

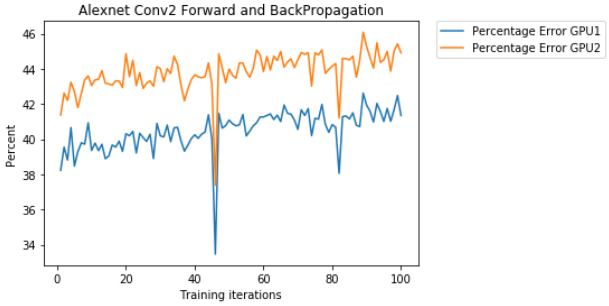

Figure 4-25 Alexnet-128 Percentage error

4.4.3 VGG batch size 12

The following figures shows the per-layer time of training on VGG-11 with batch size 12.

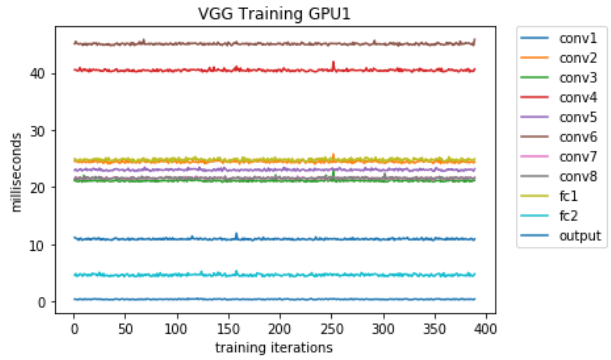

Figure 4-29 VGG Layer-Time vs number iteration GPU1

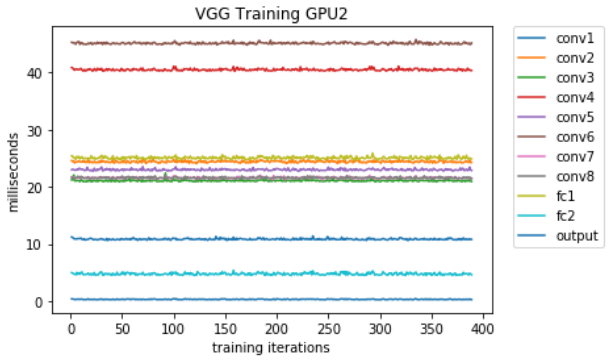

Figure 4-30 VGG Layer-Time vs number iteration GPU2

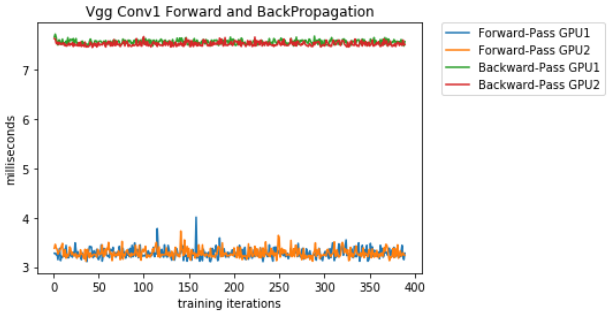

Figure 4-28 VGG Conv1 Layer-Time vs number iteration

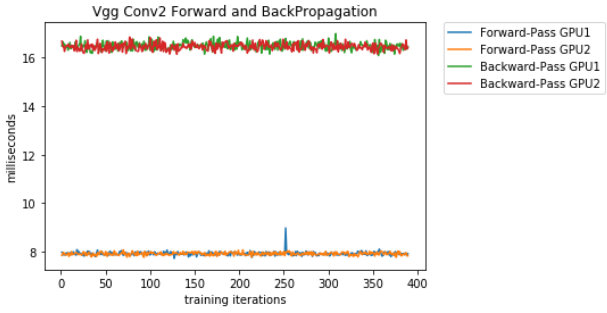

Figure 4-27 VGG Conv2 Layer-Time vs number iteration

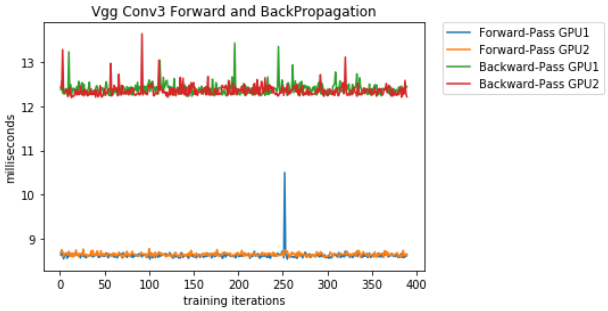

Figure 4-33 VGG Conv3 Layer-Time vs number iteration

Figure 4-32 VGG Conv4 Layer-Time vs number iteration

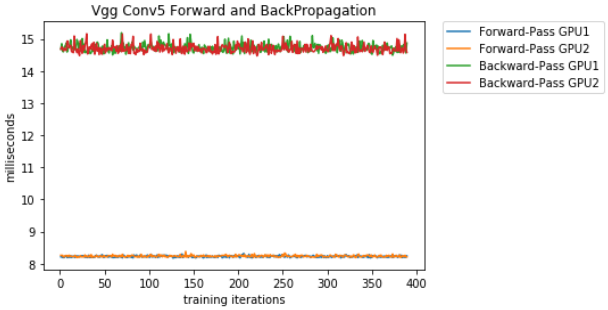

Figure 4-31 VGG Conv5 Layer-Time vs number iteration

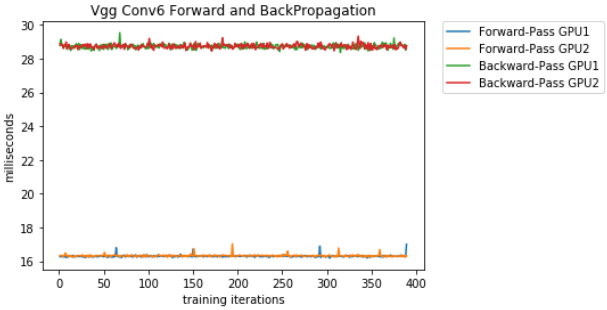

Figure 4-34 VGG Conv6 Layer-Time vs number iteration

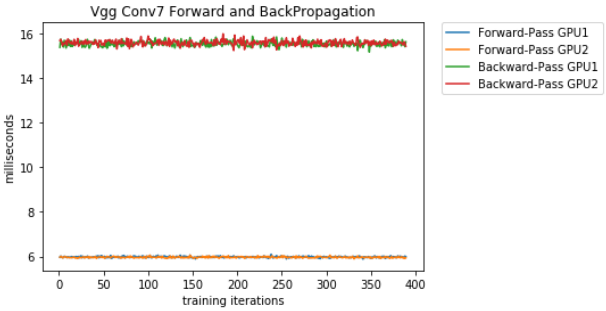

Figure 4-36 VGG Conv7 Layer-Time vs number iteration

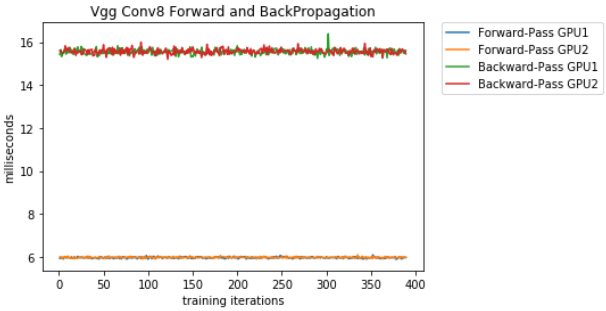

Figure 4-35 VGG Conv8 Layer-Time vs number iteration

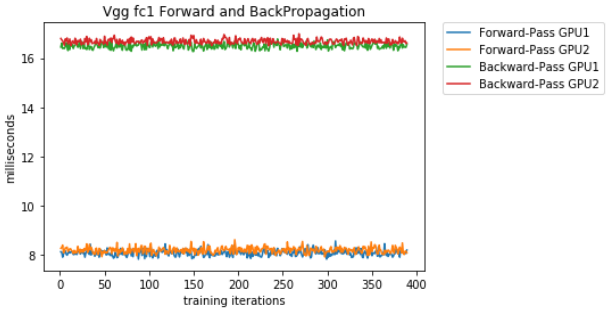

Figure 4-38 VGG fc1 Layer-Time vs number iteration

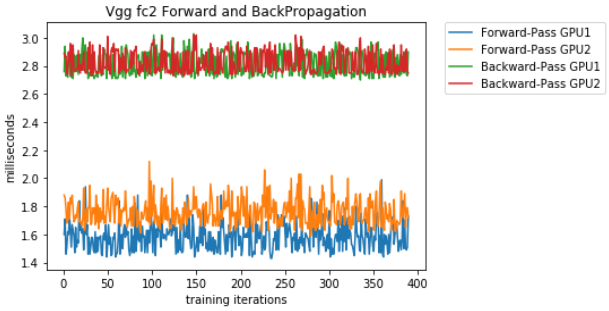

Figure 4-37 VGG fc2 Layer-Time vs number iteration

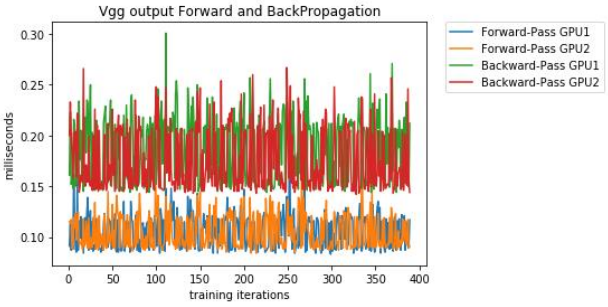

Figure 4-40 VGG output Layer-Time vs number iteration

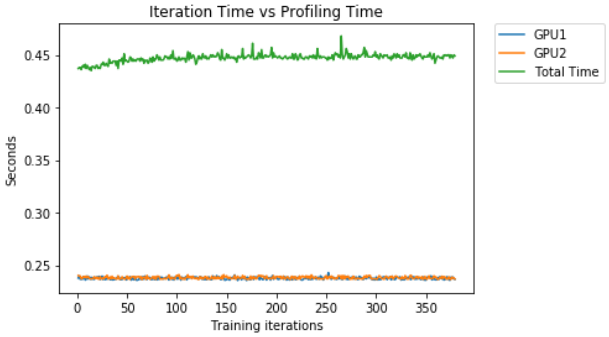

Figure 4-39 VGG Total time vs profiling time

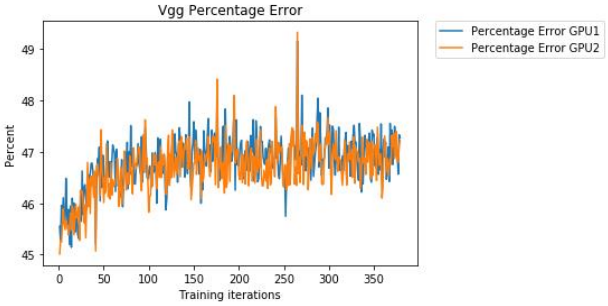

Figure 4-41 VGG Percentage Error

4.5 Analysis of Results

In this Section, we will analyze and describe the results of the test performed. First, we will describe the result of training Alexnet with batch size 64 and then training alexnet with batch size 128, finally VGG results are discussed.

4.5.1 Alexnet Batch Size 64

Inspecting the results of the test and as can be seen in the figures in the previous section, in the case of batch size 64, on GPU1 conv1 and conv2 layers takes the most time followed by conv3,conv4 which are also very close to each other with respect to computation time, then conv5, fc1, fc2 each takes less time in that order with conv5 and fc1 being a bit closer to each other in respect to computation time, output layer takes the least time followed by fc2 layer.

In case of GPU2, the times are almost identical with a minute difference here and there with the only big difference being conv5 layer falls in the range of computation time similarly with conv3 and conv4 layer.

Looking at each layer time with respect to forward pass and backpropagation, as can be seen in the figures above, Forward pass always takes less time than backpropagation. This is the same throughout all the layers apart from the output layer where some overlapping of computation time can be seen in the figure above.

The profiling output is processed and the time for each layer is extracted from it, that time is then compared with the computed time of each iteration training. Comparison of the profiling and iteration time can be seen in the figure 4.13, while figure 4.14 shows the percentage error of the time computed from the profiling. Which ranges between 44 and 54 percent. Which tells us in each iteration the time for computing these layers.

4.5.2 Alexnet Batch Size 128

Observing the result of the experiment with batch size 128, we can see in the figure 4.15 the time of different layers on GPU1, in this experiment, the time difference between conv1 and conv2 layer is quite visible, and there is a clear gap between the computation time of each layer, conv1 taking the most time followed by conv2, conv3 and conv4 are quite close in this case also with respect to computation time, then conv5, fc1, fc2 takes lesser time in the same manner as they are described. Output layer being the one taking the least time during the training process.

The computation time of each layer in GPU2 is quite like the one in GPU1 but the only difference being conv5 is a bit closer to conv3 and conv4 as was also noticed in the case of batch size 64.

Inspecting individual layer, we can see that in this experiment of batch size 128, the time taken by forward pass is always less than the time taken by backpropagation.

The result of comparing profiling time with iteration time we can see that the percentage error falls in the range between 38 and 44%.

4.5.3 VGG

In the experiment test of vgg-11, upon observing the results, in figure 4.29 the layer time vs iteration on GPU1 can be seen, the layer taking the most time is conv6 followed by conv4, fc1 and conv2 taking almost the same time, conv5 lying between the time range of fc1 and conv3, and conv3, conv7, conv8 taking almost the same time. Conv1 falls in the time range bellow conv3, conv1 is followed by fc2 and output in the same manner.

The computation time of layers on GPU2 is like GPU1 with small difference throughout all the iterations, which can be seen in figure 4.30.

The computation time of forward pass and backpropagation can be observed in the figures, the time taken by forward pass is less than backpropagation in all layers. Time of computation of output layer, in which the difference between forward pass and backpropagation is very less.

Comparing the iteration time result with the time computed from profiling output files we get the percentage error falling in the range of 45-49%.

4.6 Analysis of the approach

Upon inspecting the results and looking at the percentage error in the tests of Alexnet and VGG the percentage error lies between 38-54. which jeopardizes this approach. There is a huge difference between the iteration times and the time computed from the profiling, this can point into two possible reason for the missing time 1) the time of mem-copy is not included in this time, since profiler does not provide us that information, and 2) the GPU is not fully utilized, and that there is some idle time in the iteration time. In the test of Alexnet increasing the batch size, we saw some decrease in the percentage error. Which also points to this that, with small batch size GPU is not fully utilized.

Chapter 5

5 Conclusion

This thesis briefly described Cloud computing, Neural Networks, and its evolution into deep learning models. It highlights the basic component of a deep learning models classifier. It describes a solution for training Imagenet with different architectures and to extract the per layer computation time of a model.
Practical experiments with Convolutional Neural Network models were conducted, Proposed models were implemented using the TensorFlow library. The implemented ImageNet training application is aided by parallel computing platform CUDA, with support for multiple GPU usage in the training process.

Tests were done on Alexnet and VGG with different batch sizes. The result of the layer by layer time is compared between the iteration time and the profiling time. Comparing both times, we can see a percentage error of 38-54%. There is missing time un accounted for, which jeopardizes this approach.

Future work can include¸ building upon the outcome of this work, the per layer result can be used to develop performance models.

Bibliography

[1] Dan C. Marinescu. Cloud Computing Theory and Practice

[2] Peter Mell (NIST), Tim Grance (NIST), The NIST Definition of Cloud Computing https://csrc.nist.gov/publications/detail/sp/800-145/final

[3] Cloud Computing Paradigm Chart https://ornot.ca/2009/08/04/cloud-computing-paradigm-chart/

[4] Essentials of Cloud Computing, K Chandrasekaran

[5] Q. Zhang, L. Cheng, and R. Boutaba. "Cloud Computing: State-ofthe-Art and Research Challenges". In: *J. Internet Services and Applications* 1.1 (2010), pp. 7–18. doi: 10.1007/s13174-010-0007-6.

[6] Tensorflow vs R: A Comparative Study https://www.researchgate.net/publication/322007170_Tensorflow_vs_R_A_Comparative_Study_of_Usability

[7]: TensorFlow: Biology's Gateway to Deep Learning? https://www.ncbi.nlm.nih.gov/pubmed/27136685

[8]: Gentle introduction to Neural Networks. https://towardsdatascience.com/a-gentle-introduction-to-neural-networks-series-part-1-2b90b87795bc

[9] Neural Network Primer: Part I" by Maureen Caudill, AI Expert, Feb. 1989 https://dl.acm.org/citation.cfm?id=38295

[10] Why are GPU's necessary for Deep Learning
https://www.analyticsvidhya.com/blog/2017/05/gpus-necessary-for-deep-learning/

[11] Machine Learning it's all about GPU's
https://www.forbes.com/sites/forbestechcouncil/2017/12/01/for-machine-learning-its-all-about-gpus/#18b835e57699

[12] Why GPU's are ideal for Deep learning
https://themerkle.com/why-gpus-are-ideal-for-deep-learning/

[13] why are GPU's well suited for Deep learning
https://www.quora.com/Why-are-GPUs-well-suited-to-deep-learning

[14] Nvidia Titan-V
https://www.nvidia.com/en-us/titan/titan-v/

[15] Nvidia Tesla V100
https://www.nvidia.com/en-us/data-center/tesla-v100/

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with
deep convolutional neural networks". In: Advances in neural information processing systems. 2012, pp. 1097–1105.

[17] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning.
http://www.deeplearningbook.org MIT Press, 2016.

[18] Yann N Dauphin et al. "Identifying and attacking the saddle point problem in highdimensional
non-convex optimization". In: Advances in neural information processing systems. 2014, pp. 2933–2941.

[19] Richard S Sutton. "Two problems with backpropagation and other steepest-descent learning procedures for networks". In: Proc. 8th annual conf. cognitive science society. Erlbaum. 1986, pp. 823–831.

[20] Ning Qian. "On the momentum term in gradient descent learning algorithms". In: Neural
networks 12.1 (1999), pp. 145–151.

[21] Martín Abadi et al. f: Large-Scale Machine Learning on Heterogeneous Systems. Software available from tensorflow.org. 2015. url: https://www.tensorflow.org/

[22] Vinod Nair and Geoffrey E Hinton. “Rectified linear units improve restricted boltzmann
machines”. In: Proceedings of the 27th international conference on machine learning (ICML-10). 2010, pp. 807–814.

[23] Nitish Srivastava et al. “Dropout: a simple way to prevent neural networks from overfitting.” In: Journal of machine learning research 15.1 (2014), pp. 1929–1958.

[24] Olga Russakovsky et al. “ImageNet Large Scale Visual Recognition Challenge”. In: International Journal of Computer Vision (IJCV) 115.3 (2015), pp. 211–252. doi: 10.1007/s11263-015-0816-y.

[25] GoogleNet: Christian Szegedy et al. “Going deeper with convolutions”. In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2015, pp. 1–9.

[26] VGG: Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for largescale image recognition”. In: arXiv preprint arXiv:1409.1556 (2014).

[27] John Duchi, Elad Hazan, and Yoram Singer. “Adaptive subgradient methods for online
learning and stochastic optimization”. In: Journal of Machine Learning Research 12.Jul (2011), pp. 2121–2159.

[28] Matthew D Zeiler. “ADADELTA: an adaptive learning rate method”. In: arXiv preprint
arXiv:1212.5701 (2012).

[29] Nvidia Quadro P600 https://www.pny.com/nvidia-quadro-p600

[30] Intel Xeon Silver 4114 https://ark.intel.com/products/123550/Intel-Xeon-Silver-4114-Processor-13_75M-Cache-2_20-GHz

[31] Alexnet Architecture http://amitkushwaha.co.in/image-aesthetic-assessment.html

[32] Image convolution
http://machinelearninguru.com/computer_vision/basics/convolution/image_convolution_1.html

[33] Google net architecture graph
http://joelouismarino.github.io/blog_posts/blog_googlenet_keras.html

[34] VGG Net Architecture graph
https://medium.com/@sidereal/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5

[35] K-means http://stanford.edu/~cpiech/cs221/handouts/kmeans.html

[36] Apriori Algorithm
https://rasbt.github.io/mlxtend/user_guide/frequent_patterns/apriori/

[37] Atmosphere Project https://www.atmosphere-eubrazil.eu/

[38] Theano http://deeplearning.net/software/theano/

[39] Lasagne http://lasagne.readthedocs.io/en/latest/

[40] Blocks https://github.com/mila-udem/blocks

[41] Keras https://keras.io