Performance Analysis of MXNet for Image Classification

HARPRAGAAS SINGH

SUPERVISED BY: DR. JOSH MILTHROPE

THE AUSTRALIAN NATIONAL UNIVERSITY

OCTOBER 2019

Except where otherwise indicated, this report is my own original work.

Harpragaas Singh

25th October 2019

ACKNOWLEDGEMENTS

I would like to thank my supervisor Dr Josh Milthorpe, for gladly devoting so much of his time to teaching and helping me develop my ideas during the past two semesters.

To my dad Ramneek Singh, my mom Baljit Kaur and my sister Elahi Arora for always believing in me whenever I doubted my own capabilities.

To my friends Manchit Nathial and Abhinav Pandey for all the confidence, jokes and encouragement throughout the process.

ABSTRACT

In recent times, the application of deep learning has become widespread and has achieved great success in many AI applications. Pytorch, MXNet, TensorFlow etc. are popular distributed deep learning frameworks used for these applications. Each deep learning framework performs differently based on their hardware configuration.

The study focuses on image classification since it is one of the most popular deep learning applications. The study aims on conducting performance analysis for MXNet and compare its performance with that of the Pytorch Model. The study demonstrates performance differences between these two deep learning frameworks and understand the claims made by MXNet developers on why MXNet is a unique choice amongst the state-of-the-art deep learning frameworks.

The study concludes that MXNet trained faster with respect to PyTorch. The study accesses this conclusion by performing benchmarking on two different models and four key benchmarking metrics.

Contents

List of Figures

1. INTRODUCTION

In the last few years, the impact of deep learning has been widespread from healthcare to manufacturing. Deep learning is an artificial intelligence function that imitates the working of the human brain in processing data and creating patterns for use in decision making. Deep learning frameworks have been widely used for deep learning applications in both academia and industry. Benchmarking is an important process for obtaining a measure to determine the difference between different approaches and not focus on determining whether the approach used is better or worse than the other.
The recent popularity of deep learning frameworks has generated considerable interest in performance analysis of the deep learning frameworks. The study aims on doing a performance analysis for MXNet and comparing its speed with MXNet's direct competitor Pytorch.

1.1. Report Outline

Throughout this report, the study discusses underpinning concepts such as image classification, CNN models and provide an overview of these concepts in Section 2. The study discusses the dataset, architecture of the models on which the performance analysis is based in Section 3. In Section 4, the study provides the performance results for the benchmark measures described in the previous section. The study concludes in Section 5.

2. BACKGROUND AND RELATED WORK

2.1. Background

2.1.1. Deep Learning

Deep learning is a machine learning technique that teaches computer to learn by experience and acquire skills without human involvement. Deep learning algorithms are inspired by the working of the human brain. The deep learning algorithm like humans learns from examples by repeatedly performing the task. Over the last few years advances in the field of Deep Learning has made tasks such as speech recognition and image classification possible [1] [LeCun et al. 2015].

2.1.2. Image Classification

The task of predicting what an image represents is called Image Classification. Computers are unable to understand images as humans do; but computers can perform calculations on numbers. Thus, the images are converted into numbers for the computers to understand them. Image classification and recognition holds potential for a wide range of uses and industries from security to product discoverability to marketing.
The traditional approach used for image classification was based on feature extraction. Feature extraction involves extracting a high level of information using techniques like traditional hand engineering feature extraction, from the training set images. Then the features are trained using classification modules like SVM, Random Forest and Logistic Regression.
There are variety of challenges that are faced by the traditional approach used by Image Classification task. Since the features are extracted in an unsupervised manner, that is, the classes of the image have nothing to do with the information extracted. Other challenges faced by image recognition includes problems such as image deformation, background clutter, viewpoint variation, scale variation when testing on a set of images.
Deep Learning networks used for image classification in this study are Convolutional Neural Networks, the basic idea is that the model takes an RGB image and perform a series of transformations on the image. On each transformation the model learns a denser representation of the image. Then the model applies transformation on this dense representation, learning more abstract features of the image. At the end, these features are used to perform image classification and performs surprisingly well. [2] [Lee 2015].

2.1.3. Convolutional Neural Network

In recent years, convolutional neural network(CNN) has demonstrated excellent achievement in image classification, thus solving the above-mentioned problems. CNN is composed of single or multiple blocks of convolution and sub-sampling layers, followed by one or more fully connected layers and an output layer as shown in Fig 2.1. [3] [Sultana et al.2019].
The idea behind CNN is to reduce the image into a form which is easier to transform the input to extract higher-level features which are critical for getting a good prediction.

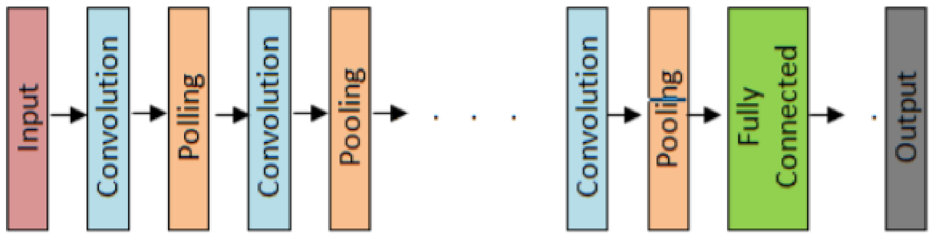


Fig 2.1 Building Block of a CNN [3]

I. Convolutional Layer

Convolutional layer is the first layer to extract features from an input image. Image features are learned using small square of input data. This small section passing over the image is called filter (Kernel) as seen in Fig 2.2. Convolution of an image with different kernels can perform operations such as edge detection, blur and sharpen by applying filters. [3][Sultana et al.2019]

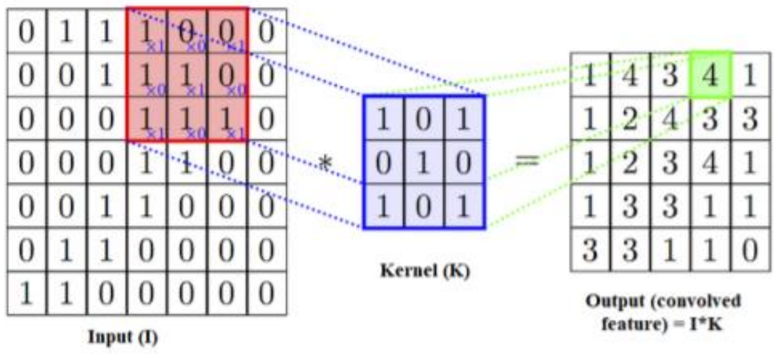


Fig 2.2 Convolutional Layer [3]

II. Pooling Layer

Pooling simply means down sampling of an image. It takes small region of the convolutional output as input, down samples it to produce a single output. Pooling are of three different types: Max Pooling take the largest element from the feature map as shown in Fig 2.3. Pooling reduces the number of parameters to be computed but makes the network invariant to translations in shape, size and scale. [3][Sultana et al.2019]

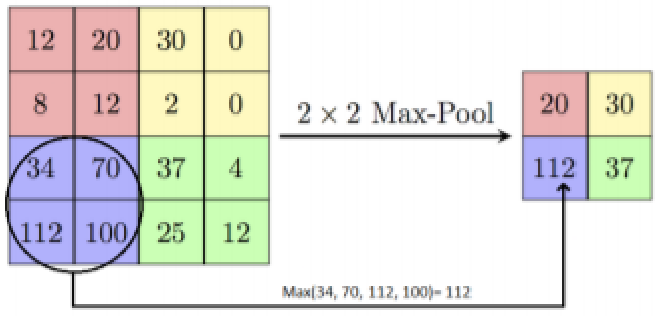


Fig 2.3 Max Pooling operation [3]

III. Fully Connected Layer (FC Layer)

The FC Layer takes the input from all neurons in the previous layer and performs operation with individual neuron in the current layer to generate output, that is, this layer flattened the matrix into vector and feed into a fully connected layer like neural network [3] [Sultana et al.2019].With FC layers all these features are combined to create a model. The outputs generated are then classified using activation functions such as SoftMax, sigmoid or ReLu, as discussed in Section 2.1.3.

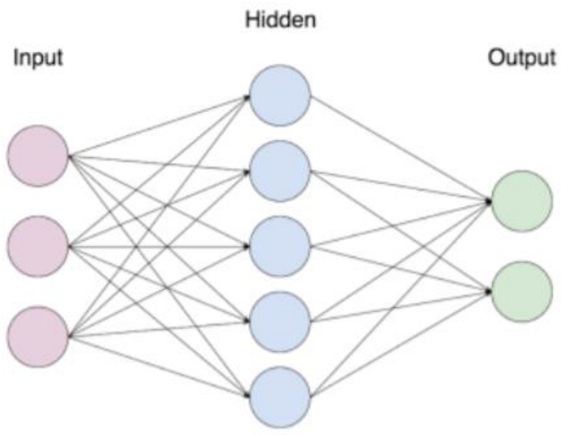


Fig 2.4 Fully connected Layer [3]

2.1.4. Different Models of CNN for Image Classification

The traditional models were built on the fact that increasing the depth of the network will also increase the accuracy of the network, provided there is no over-fitting. The problem with this approach is that the earlier layers are almost negligible learned. This is called vanishing gradient. The second problem with this approach is that by adding multiple layers will lead to higher training error thus effecting the models' performance. This is called the degradation problem. [4]

I. AlexNet

This architecture was one of the first CNN models which showed significant improvement in image classification in comparison to traditional methodologies, designed by Krizhevky et al.(2012) [5]. The model is composed of 8 trainable layers. Among them 5 are convolutional layers followed by 3 fully connected layers. AlexNet uses Rectified Linear Unit (ReLu) for the non-linear part instead of Tanh or Sigmoid function that were used in traditional CNN models like LeNet. The reason for using ReLu is to solve the vanishing gradient problem faced by the previous model. [3][Sultana et al.2019]

This model also uses the concept of dropout layer which reduces the problem of over-fitting. As per Hinton et al. [6] the concept of dropout helps in reducing overfitting. The other view of looking at dropout is that it adds noise to the network and a network trained with dropout is more robust to noise, making the model more robust and drive it towards creating useful features.

II. VGGNet

This model was proposed by Simonyan and Zisserman (2014) [7]. It makes the improvement over AlexNet by using a deeper configuration of AlexNet.

In the VGG model the convolutional layers are followed by 3 fully connected layers. The width of the networks starts at a small value of 64 and increases by a factor of 2 after every pooling layer

This model replaces large kernel-sized filters with multiple 3x3 kernel-sized filters one after another. The use of such multiple small filters showed better performance with respect to the previous models. This is because multiple layers increase the depth of the network which enables it to learn more complex features. [3][Sultana et al.2019]

III. Residual Network

He et al. (2016) [4] experienced the vanishing gradient and problems when dealing with deeper and multilayer CNNs. The authors proposed a deep residual learning framework that uses residual mapping instead of underlying mapping into their framework into their model and named it as ResNet.

The architecture is similar to the VGG model consisting of mostly 3x3 filters. From the VGG model, shortcut connection is used to form a residual network which learns the features on top of already available input. This model has achieved better accuracy and is also computationally more efficient than the VGG model. [3][Sultana et al.2019]

2.1.5. Deep Learning Frameworks

A deep learning framework is an interface which allows us to build deep learning models more easily and quickly. They provide a clear and concise way for defining models using a collection of pre-built and optimized components. Deep learning frameworks offer building blocks for designing, training and testing deep neural networks through a high-level programming interface.

I. MXNet

MXNet is a multi-language machine learning library to ease the development of machine learning algorithms, especially for deep neural networks. MXNet is computation and memory efficient. MXNet works on various heterogenous systems, ranging from mobile to distributed GPU clusters.

MXNet supports multiple languages like C++, Python, R, Julia etc. This eliminates the need for learning a new language to use the framework and simplify network definitions. MXNet models are portable in a manner such that they are able to fit in very small amounts of memory. MXNet can also scale to multiple GPUs and machines. Thus, making MXNet programmable, portable and scalable. [8]

II. Pytorch

Pytorch is a python machine learning package based on Torch, which is an open-source machine learning package based on programming language Lua. Pytorch is based on tensor computation with strong GPU acceleration.

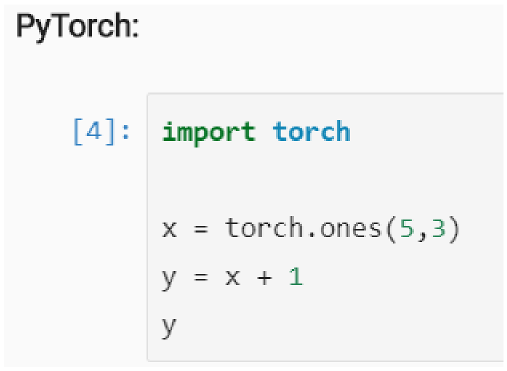
Pytorch modelling process is simple and transparent because of the framework's architectural style. Pytorch features a lot of pretrained models and modular parts that are ready and easy to combine. Pytorch is similar to traditional programming and uses common debugging tools. [9]

- Comparison between Pytorch and MXNet [10]

Pytorch is a popular deep learning framework due to it's easy to understand API and its completely imperative approach. MXNet gives you the simplicity and flexibility of Pytorch and allows you to hybridize the network.

A. Data Manipulation

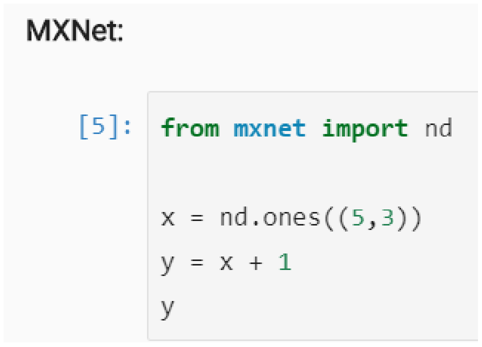
Pytorch and MXNet relies on multidimensional matrices. Pytorch follows torch's naming conventions and call these matrices as "tensors", MXNet on the other hand follows NumPy's convention and refers to them as "NDArrays" as shown in Fig 2.5



PyTorch:

```
[4]: import torch

     x = torch.ones(5,3)
     y = x + 1
     y
```



MXNet:

```
[5]: from mxnet import nd

     x = nd.ones((5,3))
     y = x + 1
     y
```

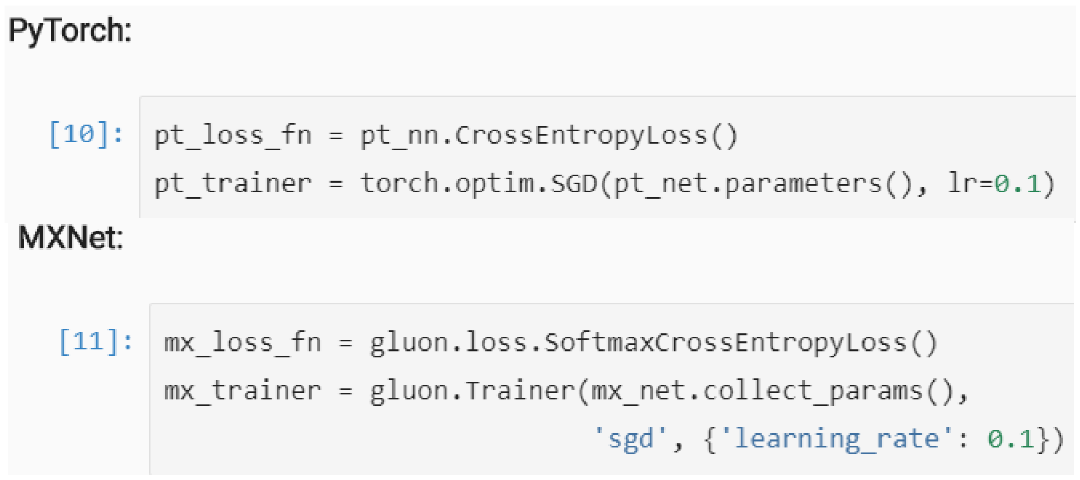
Fig 2.5 Sample code snippet for data manipulation in PyTorch and MXNet [10]

B. Model Training

Pytorch and MXNet allows to download the dataset from their sources. Pytorch allows you to specify input size as the first argument of the linear object. MXNet provides extra flexibility to network structure by automatically inferring the input size after first forward pass. MXNet specifies activation functions directly in fully connected and convolutional layers.

C. Loss function and Optimization Algorithm

The difference between the two frameworks is small. The main difference is that MXNet uses a trainer class, which accepts optimization algorithms as an argument as depicted in Fig 2.6



PyTorch:

```
[10]: pt_loss_fn = pt_nn.CrossEntropyLoss()
      pt_trainer = torch.optim.SGD(pt_net.parameters(), lr=0.1)
```

MXNet:

```
[11]: mx_loss_fn = gluon.loss.SoftmaxCrossEntropyLoss()
      mx_trainer = gluon.Trainer(mx_net.collect_params(),
                                 'sgd', {'learning_rate': 0.1})
```

Fig 2.6 Sample code snippets for Loss function and optimization algorithm [10]

D. Training

MXNet in comparison to Pytorch doesn't need to flatten 4-D input into 2-D when feeding data into forward pass. In MXNet the calculations are performed within the autograd.record() scope so that it can be automatically differentiated in the backward pass. Unlike Pytorch MXNet does not require to clear the gradient every time as the new gradient is written in, not accumulated.

It can be concluded that MXNet and Pytorch have many similarities. The main difference lies in terminology and behaviour of accumulating gradients. The rest of the code is very similar.

The following two points discusses the key aspects that differentiate MXNet from other deep learning frameworks:

2.1.6. Hybridization

Popular deep learning frameworks are based on only one style of programming, for example Pytorch is based on imperative programming while TensorFlow is based on symbolic programming.
MXNet tries to harness the best of both worlds. MXNet tries to harness advantages of both styles of programming. The developers believed that the users should be able to develop and debug using imperative programming while having the ability to convert it into symbolic programming. This is achieved using the Gluon API. [11]

2.1.7. Performance

The developers have demonstrated the computing performance improvement gained using hybridization. They compare the computation time before and after calling the hybridization function.
The developers time 100 net model computations that are based on imperative and symbolic programming, respectively, before and after network has called the hybridization function. The result observed showed improvement using symbolic programming. [11].

2.1.8. GPU for Deep Learning – CPU vs GPU

Graphics Processing Unit (GPU) is considered as heart of Deep Learning. It is a single chip processor used for extensive graphical and mathematical computations which frees up CPU cycles for other jobs.

CPU	GPU
Has complicated cores which run processes sequentially with few threads at a time	Has large number of simple cores which allow parallel computing through thousands of threads computing at a time
Runs the host code	Runs CUDA code
Assigns complex tasks.	Performs the complex tasks like 3D Graphics Rendering, vector computations
Are latency(memory access time) optimized	Are bandwidth optimized

Table 2.1 Comparison between CPU and GPU

Memory Bandwidth is one of the main reasons why GPUs are faster for computing than CPUs. CPU takes up a lot of memory while training the model because the models are trained on large datasets. GPU comes with VRAM memory. Thus, making GPU the ideal choice of training deep learning model.

Training deep learning models require large chunks of memory, GPUs provide the best memory bandwidth while having no drawback due to latency vis thread parallelism. For example, if your system is using thread parallelism and have task that is using large chunks of memory then the system will wait for the first GPU core, but after that the system need not wait since the task that is going on other GPU cores will queue up, thus the system will have direct access to the memory. This effectively hides latency while GPU offers high bandwidth.

However, optimizing the tasks are far easier in CPU than in GPU. This is because of the fewer yet powerful CPU cores.

CPU can be used for training smaller networks with a small dataset. However, for training a large network with large dataset, GPU is used. Also, the power cost of GPU is higher than that of the CPU.

From the above factors, CPU can be used to train the deep learning model quite slowly while GPU accelerates the training of the model. Due to the high bandwidth, hiding the latency under the thread parallelism and easily programable registers make GPU a lot faster than a CPU. Thus, GPU is a better choice to train deep learning model efficiently and effectively.

2.2. Related Work

Benchmarking is an important process for obtaining a measure to determine the difference between different approaches and not focusing on determining whether the approach used is better or worse than the other. Benchmarking is important to understand which tool will add more value and helps in understanding performance gaps between the tools.
In recent time, machine learning algorithms have become more widespread. Since the algorithms run on standard multicore processor and GPUs it is important to benchmark and do a performance analysis for the same. The study will benchmark the performance of the latest version of MXNet on different processors and draw out comparisons with Pytorch.
Shi et al. (2018) [12] benchmark the performance for deep learning papers is the one that is used as the base to start the research project. The paper discusses analytical and experimental analysis of four deep learning frameworks, namely, Caffe-MPI, CNTK, MXNet and TensorFlow over multi node environments. Post the experimental results, the paper explains how the frameworks can be further optimized and discusses about the comparison between the performances of each framework

The paper gives a good comparison for all deep learning frameworks, using different models and datasets which can be implemented using MXNet to further draw out comparisons with state-of-the-art software tools.

The paper uses MXNet *version 0.93* for benchmarking the performance while a *version 1.5.1* has been released. The latest version of MXNet offers flexible and efficient GPU computing which could potentially be an important factor in changing and comparing MXNet's performance.

2.3. Contribution of the Report

Most of the deep learning frameworks are difficult to debug due to its imperative environment while MXNet offers both low level-control and high-level API as discussed in Section 2.1.5.1. However, there has not been proper benchmarking for MXNet newer versions.
The base paper for the study uses AlexNet for a comparative analysis of MXNet. This study will train both ResNet and VGG Models, considered to be better than AlexNet [3], for image classification. The study aims on using the latest version for MXNet and Pytorch to perform a performance analysis for the two frameworks that are considered to be direct rivals in the research community.

3. METHODOLOGY

3.1. EXPERIMENTAL SETUP

3.1.1. SYSTEM CONFIGURATION

Architecture:	x86_64
CPU(s):	32
Model name:	Intel® Xeon® Gold 6134 CPU @ 3.20GHz
CPU family:	6
CPU MHz:	1200.335
CUDA Version:	10.1
GPU:	Tesla P100
MXNet Version	1.5.1
PyTorch Version	1.3.0

The operating system in use was Ubuntu 18.04.3 LTS. Both the models are built using jupyter notebook. Both deep learning frameworks used during the time of training is mentioned above. The benchmark is performed on Intel® Xeon® Gold 6134 CPU @ 3.20GHz.

Tesla P100 is used for GPU training of the model.

GPU Specifications :

GPU Architecture	NVIDIA Pascal
NVIDIA CUDA® Cores	3584
Max Power Consumption	250 W
ECC	Yes

3.1.2. DATASET

The dataset used for training the model is CIFAR-10. The CIFAR-10 are labelled subsets of 80 million tiny images dataset. They were collected by Alex Krizhevsky, Vinod Nair and Geoffrey Hinton. The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes with 6000 images per class. There are 50000 training images and 10000 test images. The test batch contains 1000 randomly- selected images from each class. The training batches contain the remaining images in random order. The following are the 10 classes in the dataset : *airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck.*
All the classes are mutually exclusive and have no overlap between different classes. The classes : *"automobile" and "truck"* do not have any common thing between them.

3.2. MODELS

This section describes the structure of ResNet20 and VGG Model that is used for training the Cifar10 dataset.

3.2.1. Residual Network

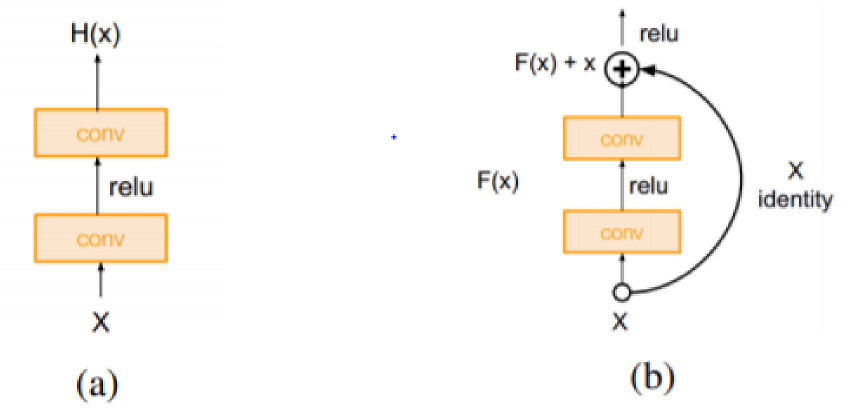


Fig 3.1 (a) Plain Layer (b) Residual Block [3]

Fig 3.1 depicts the difference between the working of plain layer and the residual block. The RESNET model replaces underlying mapping with residual mapping. [4]

Model Architecture :

The first step on the ResNet is a 3x3 convolution followed by batch normalization operation. The stride and padding are set to 1. This is done in order to keep the input size and output size the same.
For the stack of layers, the feature map sizes are {32, 16, 8} respectively with 2 convolutions for each feature map size. The number of filters for each layer is {16, 32, 64} respectively. The down sampling layer, used to reduce the dimensionality of the features, by increasing the stride to 2, for the first convolution for each layer.
For the output size to match the size of the input before the addition, the input is padded with zeros. Thus, making the bypass connection a regular *Identity Shortcut* [4] since the dimensionality is constant through the layers.
For the following two layers, that is, layer 2 and 3 behave equivalent to layer 1. The only exception being that the first convolution layer uses a stride of 2. Thus, making the output size half of the input size, with the padding of 1. This also implies that the shortcut connection will require to adjust the volume's size before summation. An average pooling operation is used to perform the down sampling right before the dense layer.

3.2.2. VGG Model

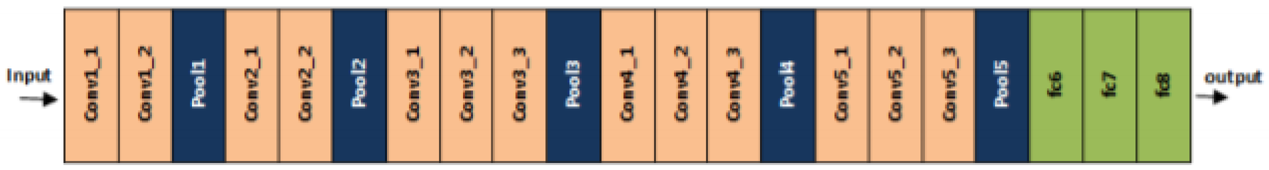


Fig 3.2 Architecture of VGGNet [3]

Fig 3.2 depicts the architecture of a VGGNet which can be manipulated depending on the use of different parameters.

Convolution layer 1 takes a fixed size of 224 x 224 image as input. Then that image is passed through a stack of convolutional layers. The filters are used with a receptive field of 3x3. The convolution stride is fixed to 1 pixel, this is to keep the spatial resolution preserved after convolution. Pooling is carried out by five maximum pooling layers, which follow some of the convolutional layers. Maximum pooling is performed over a 2x2 pixel window, where stride is 2.
Three Fully Connected layers follow a stack of convolutional layers, that is around 50 layers for the model used in this study. The configuration of the fully connected layers is the same. Hidden layers in the network are trained with ReLU. All the layers are followed by soft-max layer as the final layer.

Shi et al. (2017) [12] uses AlexNet to benchmark the performance of deep learning frameworks. However, as discussed in section 2.1.4 we know that AlexNet is one of the earlier successful CNN models. This study will perform benchmarking using the VGG and ResNet model that have proven to be better than AlexNet.

3.3. Benchmarking Measures

This section will focus on the methods that are taken into consideration for benchmarking the performance of the models. For comparison purpose, the models used for both MXNet and Pytorch uses the same CNN models for performing image classification on the Cifar-10 dataset.

The models are coded using python in jupyter notebook for the purpose of easy comparison of the structure of the models.

This section will also discuss about the Test Accuracy achieved by both the models. However, accuracies are compared just to have a sanity check on the working of the models. Hence, it is not an important measure while performing a benchmark based on the speed and processor utilization done by the models.

The benchmarking measures discusses in this section are only for image classification. However, these measures can also be extended to evaluate other deep learning algorithms.

I. Time to train the model

Training machine learning model is one of the most important measure that is used to improve performance of the model. However, it is a time-consuming step. Different deep learning frameworks shows different training time depending on the hardware of the environment it is trained in. Comparing the time taken by models to train would provide an ideal benchmarking measure as it will serve as a good reference point to the users for choosing an appropriate framework for image classification.

Thus, benchmarking the training time for a model will provide valuable information about how quick a model can be trained for a new dataset and provide substantial information about it.

II. Images trained per second

Images trained per second means that from the dataset of 60000 images and a test set images, the model correctly identifies the images from the testing images. The purpose of this measure is to identify which framework is able to detect a greater number of images, respectively, from the testing images in the quickest way.

For the purpose of benchmarking, both models will be trained on the same hardware configuration and an arbitrary set of testing images to produce a result that is comparable. The average number of images detected correctly while training the model could be verified by the testing accuracy of the model.

III. GPU Utilization

As discussed in Section 2.1.6 we conclude that GPU enabled machines speed up the time taken by deep learning models.
Thus, GPU Utilization is an important metric when it comes to benchmarking machine learning algorithms. This metric defines how frequently GPU is utilized when training the deep learning model.

GPU Utilization (in percentage) = (Active time of the GPU core)/(Total training time of the model)

The above equation is used calculate utilization of a GPU core. Ideally, the deep learning models would try to achieve high GPU Utilization when training the model as discussed by Liu et al.(2019) [13]

IV. Memory Usage

Liu et al.(2019) [13] also discusses the importance of memory usage when training a deep learning model. Training deep learning models can be a memory-consuming, as large number of parameters, derivatives and temporary variables use the memory space.
Memory usage is an ideal benchmarking metric as it helps in observing the scalability of deep learning models. The memory consumption of a model can give information on whether training the model is feasible on a system with multiple GPUs or even a mobile device.

4. RESULTS

This section will demonstrate the performance analysis of the benchmarking measures as discussed in section 2.4. The results discussed and achieved will help to identify the performance capabilities of the deep learning frameworks with respect to image classification.

This section is divided into two sub sections discussing the results achieved using the Residual Network model and the VGG model.

4.1. RESNET20

This section covers the initial toy benchmark conducted in order to compare the performance of MXNet with Pytorch. The results achieved using this model were indicative that MXNet performs approximately 2 times faster than Pytorch. The study uses the most basic Residual Network architecture, that is, RESNET20.

The results observed acted as a point of reference for the study to carry out a performance analysis of these frameworks on a complex VGG Model.

4.1.1. Hyperparameters

This section specifies the Hyperparameters used for training RESNET20. The Hyperparameters used as specified in table 4.1 have been kept the same when training for both MXNet and Pytorch Model.

Hyperparameters	**MXNet**	**PyTorch**
Batch Size	128	128
Number of Epochs	10	10
Learning Algorithm	Adam	Adam
Learning Rate	0.001	0.001
Loss Function	SoftMax Cross Entropy Loss	Cross Entropy Loss

Table 4.1 Hyperparameters for RESNET20

The Hyperparameters specified here can be changed and manipulated using trial and error to get a better accuracy. However, as discussed good accuracy is not the main goal of this project. Thus, the hyperparameters used here are the same for both deep learning models in order to have a fair comparison.

The testing accuracy achieved on testing the model for both MXNet and Pytorch on RESNET20 is 83.19% and 80.67% respectively.
MXNet performs slightly better than the Pytorch model for this case, however concluding the fact that MXNet model will always perform better than Pytorch model is misleading since training of data depends on shuffling of data which is performed randomly.

4.1.2. Time to train the model

In section 3.4 I. we discuss the importance of the *time to train the model benchmark.* The following results are based on the hyperparameters specified in section 3.1.1

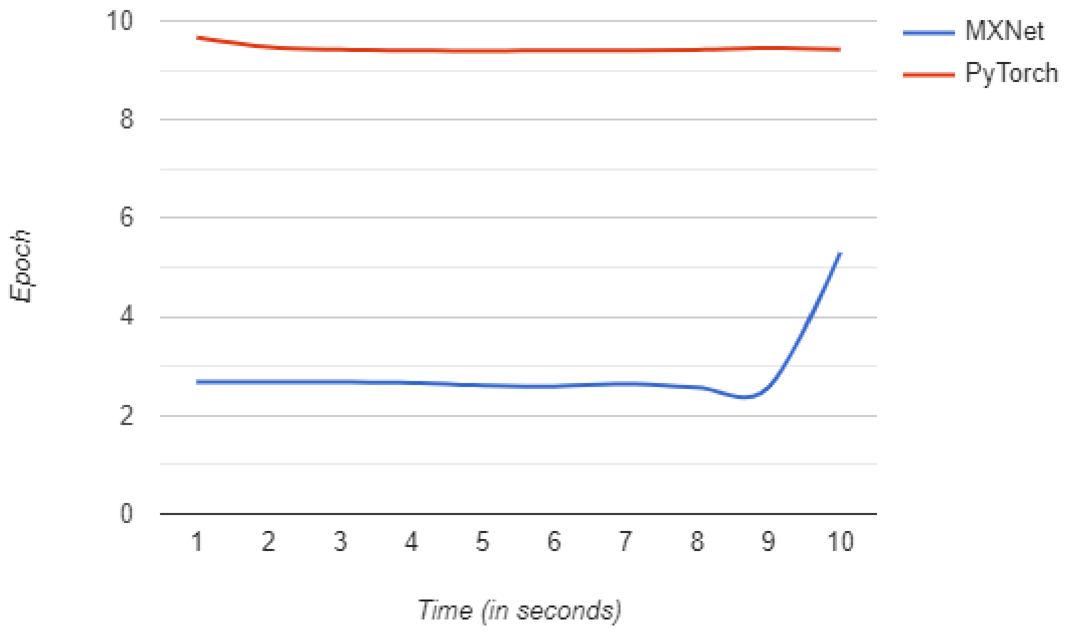


Fig 4.1 Time to train MXNet model and PyTorch model for RESNET20

Fig 4.1 depicts the time taken per epoch by the MXNet model and Pytorch Model on RESNET20 where MXNet and Pytorch is represented by *blue and red line* respectively.

It is important to note that the time used for the graph analysis may vary slightly when trying to reproduce the same results, however there will not be any significant difference with the results.

In the above graph , MXNet ranges from 2.572 seconds to 5.302 seconds while Pytorch ranges from 9.388 seconds to 9.664 seconds. The above graph clearly indicates the time difference between the two frameworks where MXNet model trains faster than the Pytorch Model. From the graph it is evident that the maximum time taken by the MXNet model to train the model per epoch is lesser than the minimum time taken by the PyTorch model.

The total time taken by the MXNet model is 29 seconds while the PyTorch model takes around 1 minute and 34 seconds.

It is important to note than even on multiple runs for both models achieved similar results with negligible difference in the time. A table describing the same is provided in Appendix Section 7.4 .

Thus, based on the above observations, we can see that MXNet model performs twice as fast as Pytorch when training the model.

4.1.3. GPU Utilization

GPU utilization is one of the primary metrics to observe during the training of a deep learning model. This metric is analysed using NVIDIA's GPU monitoring interface '*nvidia-smi*'.
Monitoring the models' GPU Utilization is one of the best indicators to determine if the GPU is being used, as discussed in Section 3.4 III.

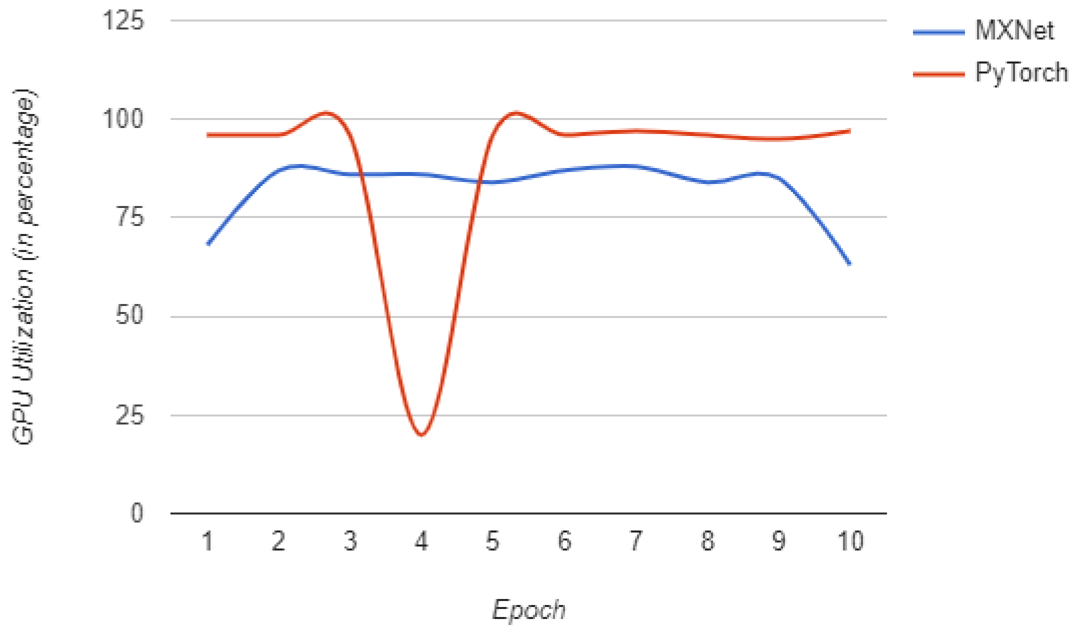


Fig 4.2 GPU Utilization by MXNet model and PyTorch model for RESNET20

From Fig 4.2, it is observed that MXNet utilizes 84% of GPU on average while the Pytorch model utilizes 97% of GPU on average.
However, on using the GPU's monitoring tool '*nvidia-smi -l 1*' that provides the GPU status every second. This is indicated by Fig 4.2.
It was observed that the GPU Utilization for training the MXNet model showed the same trend throughout the training time on the other hand Pytorch model showed a few major drops during the process.
The potential reason for this is that the Pytorch model can be spending too much time doing IO operations.
MXNet utilizes a decent amount of GPU however the percentage of GPU used with respect to PyTorch is relatively less. The most likely reason for underutilization of GPU is using a small

batch size, that is, computer spends more time loading and unloading training data than performing compute operations. Yet, training on the same parameters yields good accuracy.

Thus, because of the constant utilization, the MXNet model can be considered a favourable model in comparison to the Pytorch model.

4.1.4. Memory Usage

The memory Usage for MXNet and Pytorch model shows a similar trend, that is, the memory usage shows a sudden increase as the number of epochs increase for RESNET20. It is also observed that the memory usage showed an increase after the first epoch and remained constant for the remaining epochs. The difference between the memory usage for both the models is observed to very small.
On further analysis, Liu et al.(2019) [13] discusses the effects of batch size on memory usage.

Thus, from this observation it can be concluded that Memory Usage is not an ideal benchmarking metric for performance comparison.

4.2. VGG

The above benchmark for RESNET20 is considered as a "toy benchmark" for this project. The promising results achieved using this benchmark act as motivation for conducting a more detailed benchmark analysis for the models, using a more complex architecture that will be covered in this section.

4.2.1. Hyperparameters

This section specifies the Hyperparameters used for training VGG. The Hyperparameters used as specified in Table 4.2 have been kept the same when training for both MXNet and Pytorch Model.

Hyperparameters	**MXNet**	**PyTorch**
Batch Size	128	128
Number of Epochs	10	10
Learning Algorithm	Adam	Adam
Learning Rate	0.001	0.001
Loss Function	SoftMax Cross Entropy Loss	Cross Entropy Loss

Table 4.2 Hyperparameters for VGG

The Hyperparameters specified here can be changed and manipulated using trial and error to get a better accuracy. However, as discussed good accuracy is not the main goal of this project. Thus, the hyperparameters used here are the same for both deep learning models in order to have a fair comparison.

The testing accuracy achieved on testing the model for both MXNet and Pytorch on VGG is 82.55% and 83.03% respectively.
As opposed to the test accuracy result for RESNET20 MXNet performs slightly poor than Pytorch model for this case, however concluding the fact that Pytorch model will always perform better than Pytorch model is misleading since training of data depends on shuffling of data which is performed randomly, as discussed in 4.1.1

4.2.2. Time to train the model

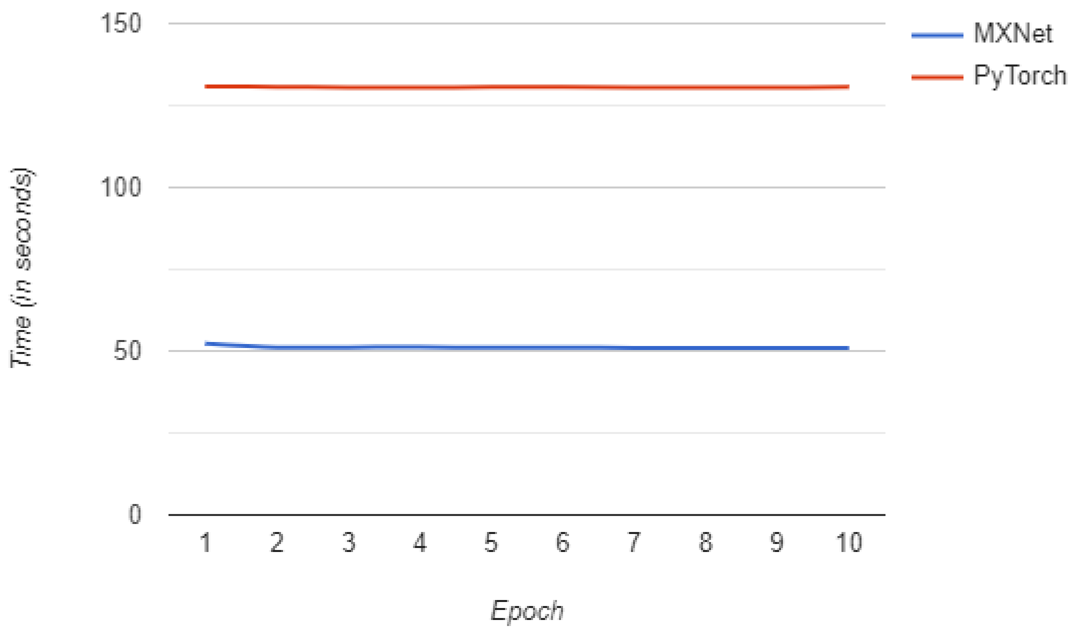


Fig 4.3 Time to train MXNet model and PyTorch model for VGG

Fig 4.3 depicts the time taken per epoch by the MXNet model and Pytorch Model on VGG where MXNet and Pytorch is represented by *blue and red line* respectively.

It is important to note that the time used for the graph analysis may vary slightly when trying to reproduce the same results, however there will not be any significant difference with the results.

In the above graph , MXNet ranges from 50.905 seconds to 52.285 seconds while Pytorch ranges from 130.45 seconds to 130.830 seconds. The above graph clearly indicates the time difference between the two frameworks where MXNet model trains faster than the Pytorch

Model. From the graph it is evident that the maximum time taken by the MXNet model to train the model per epoch is lesser than the minimum time taken by the PyTorch model.

The total time taken by the MXNet model is 8 minutes and 52 seconds while the PyTorch model takes around 22 minutes and 19 seconds.

It is important to note than even on multiple runs for both models achieved similar results with negligible difference in the time. A table describing the same is provided in Appendix Section 7.4 .

Thus, based on the above observations, we can see that MXNet model performs twice as fast as Pytorch when training the model.

4.2.3. Images trained per second

As discussed in section 3.4. II., images trained per second is one of the key metrics to benchmark performance for deep learning models.

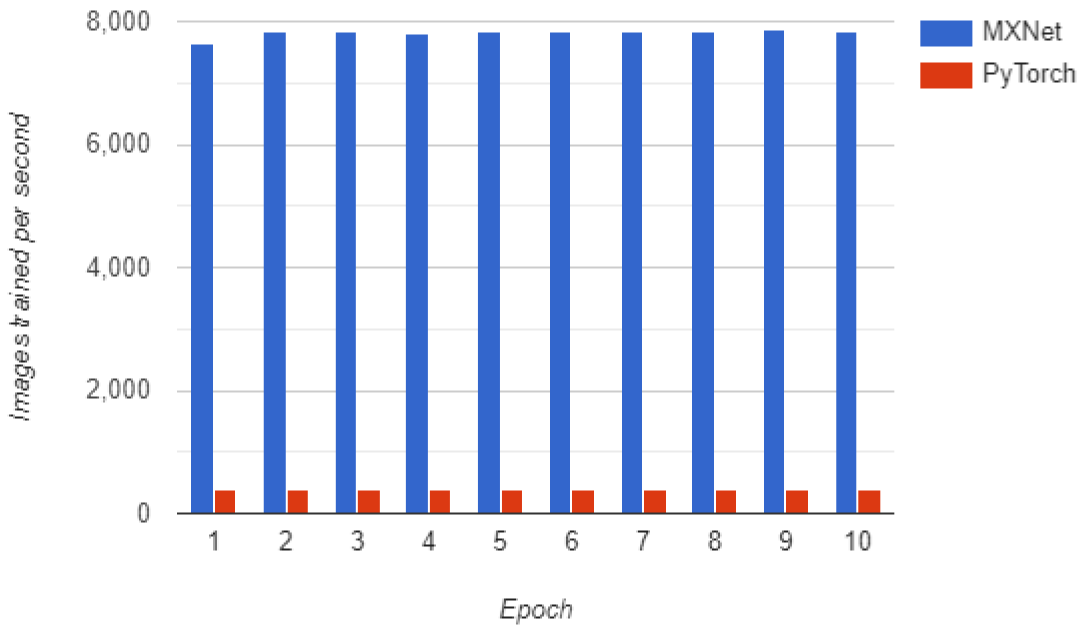


Fig 4.4 Images trained per second by MXNet model and Pytorch Model

Fig 4.4 depicts images trained per second per epoch by the MXNet model and Pytorch Model on VGG where MXNet and Pytorch is represented by *blue and red bars* respectively.

From Fig 4.4, it is observed that Images trained per second by the MXNet model is substantially high, that is, around 7800 images per second on average. On the other hand, the number of images predicted by Pytorch were around 383 images per second on average.

Thus, MXNet identifies approximately 7400 more images than Pytorch model in less amount of time as observed in section 4.2.2.

4.2.4. GPU Utilization

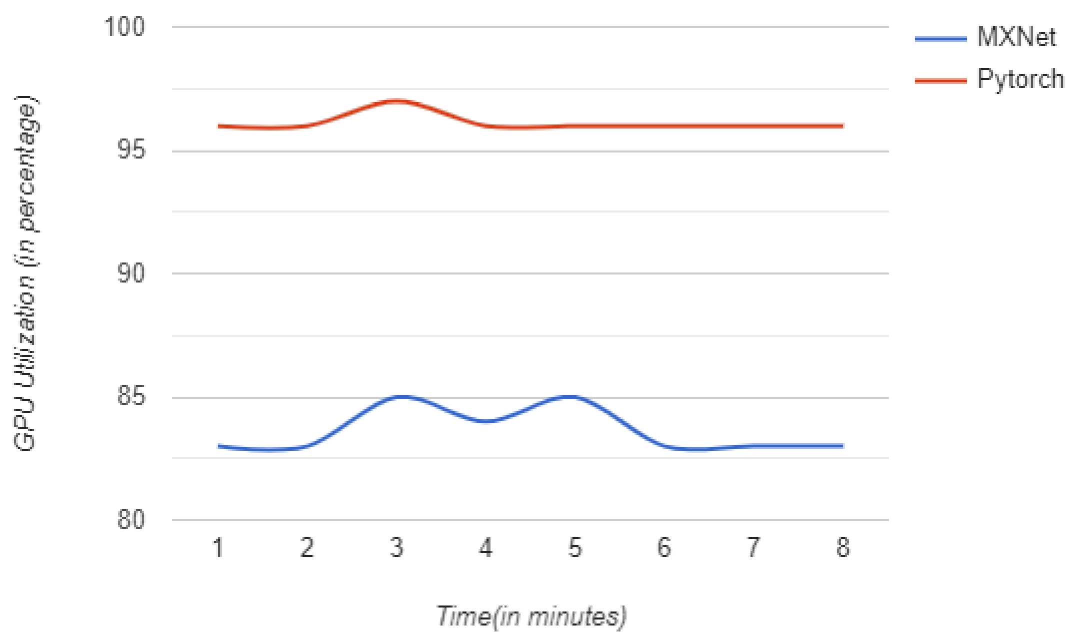


Fig 4.5 GPU Utilization by MXNet model and PyTorch model for VGG

It was observed that GPU Utilization shows the same trend for both Pytorch and MXNet for VGG architecture as it was observed for Pytorch and MXNet for RESNET 20.

Due to the complex architecture used the study uses GPU monitoring tool '*nvidia-smi -l 60*' that provides the GPU status for every 60 seconds. This is indicated by graph (graph no.).
It was observed that the GPU Utilization for training the MXNet model showed the same trend throughout the training time on the other hand Pytorch model performed better with respect to MXNet.
MXNet utilizes a decent amount of GPU however the percentage of GPU used with respect to PyTorch is relatively less. The most likely reason for underutilization of GPU is using a small batch size, that is, computer spends more time loading and unloading training data than performing compute operations. Yet, training on the same parameters yields good accuracy.

From Fig 4.5 it is observed that PyTorch model is favourable in comparison to MXNet model.

4.2.5. Memory Usage

The memory Usage for MXNet and Pytorch model shows a similar trend, that is, the memory usage shows a sudden increase as the number of epochs increase for VGG. It is also observed that the memory usage showed an increase after the first epoch and remained constant for

the remaining epochs. The difference between the memory usage for both the models is observed to very small.
This trend was similar to the observation discussed in Section 4.1.4. The only difference was the amount of memory used due to the use of different models.

4.3. Why MXNet?

MXNet offers a comprehensive and flexible Python API, Gluon API which gives you the simplicity and flexibility of Pytorch and aims to bridge the gap between imperative nature of MXNet and its symbolic capabilities and keep the advantages of both through hybridization as discussed in Section 2.1.4 '*Comparison between MXNet and PyTorch*'. MXNet developers claim that MXNet supports two styles of programming *: imperative programming and symbolic programming.*
A comparison of these two methods show that imperative programming is easier, straightforward, easy to write and at the same time easy to debug. Pytorch is based on imperative programming. Symbolic programming is more efficient and easier to port. Symbolic programming makes it easier to optimize the system and can port it into a format independent of Python. This avoids any potential performance issues related to the Python interpreter.

4.3.1. Improvement in performance

For the above two models the study uses *hybrid blocks* for the purpose of using Hybridization, calling the *hybridize* function convert imperative program into symbolic improvement. This method as discussed, improves the computation performance significantly.
This is observed from the above benchmarking analysis. MXNet perform considerably well for both the models in comparison to Pytorch.
The study believes that hybridization improves the performance of MXNet considerably. In order to prove the above claim, the study conducted a performance analysis for MXNet model using just imperative programming style and the Hybrid MXNet model:
It was observed that Hybrid MXNet model performs better with respect to the imperative programming MXNet model. The training time of the model was reduced by a small value for VGG while so significant when training on RESNET20. However, images trained per second was improved by a significant amount, that is, from training 1000 images per second to approximately 8000 images per second as observed in Fig 4.6.

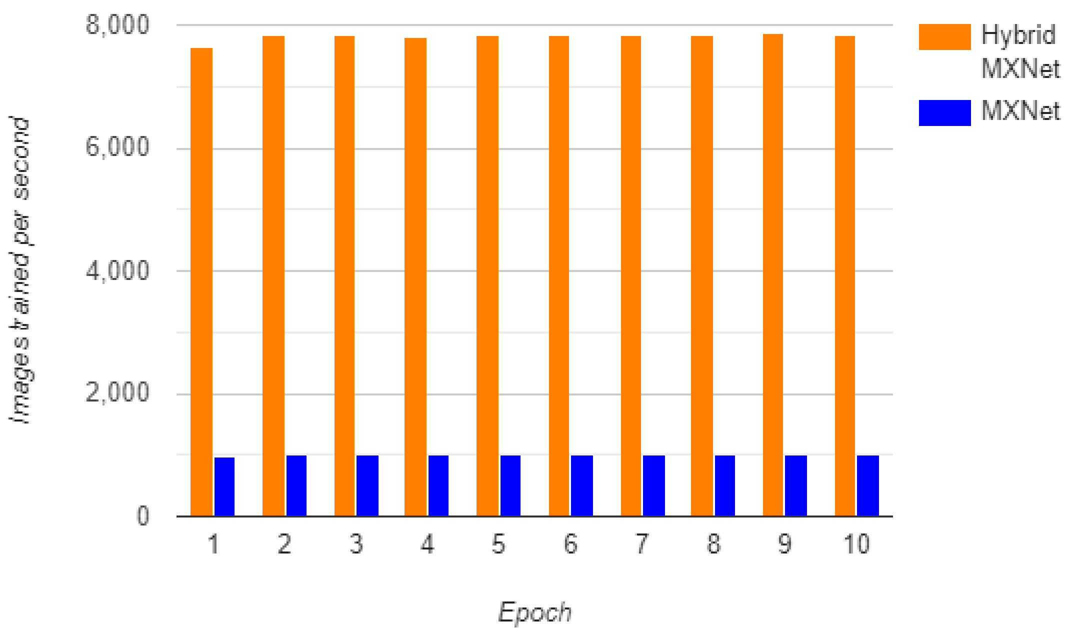


Fig 4.6 Images trained per second for Hybrid MXNet and MXNet for VGG

This means that a Hybrid MXNet model can train on a greater number of images with respect to a normal MXNet model in same amount of time resulting in a more accurate classification of image.

In terms of GPU Utilization and Memory Usage, the model showed similar trend. Thus, indicating that GPU Utilization and Memory Usage depends on the hyperparameters used to train the models.
Liu et al.(2019) [13] discusses the effect of batch size on GPU Utilization and Memory Usage.

4.4. Summary

In this new world, with bigger datasets neural networks dominate on most pattern recognition problems. Over the last few years, neural networks are dominating the field of computer vision. With GPUs accelerating neural network training, MXNet offer powerful tools to help developers exploit the full capabilities of GPUs.

This is observed from the results observed in the above sections, where MXNet has performed faster than Pytorch as discussed in section 4.3.1.

5. CONCLUSION

Until recent time, the machine learning frameworks used for deep learning have varied according to individual preference. As the field of machine learning grows, a preference-based system can have shortcomings leading to inefficiencies. As a result of which benchmarking deep learning frameworks is important.

For the purpose of performance analysis, the study benchmarks Pytorch and MXNet. These two frameworks are considered because they both are known to be quick and efficient, which allows for easy code debugging. Both frameworks also have their respective strength : Pytorch is widely used in the research community while MXNet is fast. Thus, the aim of this study is to compare the speed of MXNet and Pytorch.

From the performance analysis, the study concludes that MXNet is twice as fast as Pytorch. The reason for the same is as follows: MXNet is an ultra-scalable, flexible and deep learning framework that supports multiple languages that places a special emphasis on speeding up the development of neural networks. MXNet offers Gluon API, a powerful interface for faster machine learning model building without affecting the model's performance. Furthermore, MXNet uses hybridization, combining imperative and symbolic programming. This combines the advantages of different deep learning frameworks using the above two programming styles respectively. Such an approach is certain to improve model training.

The benchmark reflects this performance improvement as discussed in Section 4.3.1. The study concludes that given MXNet's combination of high performance and access to a high-level API, it stands out as a unique choice among deep learning frameworks.

5.1. Future Work and Improvements

During development and evaluation, the study discovered a few new directions for the performance analysis of MXNet by comparing it other deep learning frameworks like TensorFlow, Caffe. Since developers are constantly developing more features for deep learning frameworks, benchmarking them would be an ideal case study to analyse their performance. Similarly, the benchmark could be performed on more complex models.
The study also mentions a few benchmarking measures that were considered important for a performance analysis; however, it was concluded that they were dependent on the hyperparameters. A study about the effect of hyperparameters on such measures could be valuable

Bibliography

[1] Y. B. a. G. H. Yann LeCun, "Deep Learning," 2015.

[2] A. Lee, "Comparing Deep Neural Networks and Traditional Vision Algorithms in Mobile Robotics," 2015.

[3] F. S. A. S. P. Dutta, "Advancements in Image Classification using Convolutional Neural Network," 2019.

[4] X. Z. S. R. a. J. S. K. He, "Deep residual learning for image recognition," 2016.

[5] I. S. G. E. H. A. Krizhevsky, "Imagenet classification with deep convolutional neural networks," 2012.

[6] N. S. A. K. I. S. R. R. S. Geoffrey E. Hinton, "Improving neural networks by preventing co-adaptation of feature detectors," 2012.

[7] K. S. a. A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014.

[8] M. Gupta, "MXNet: What is it and How to Get Started," *https://becominghuman.ai/mxnet-what-is-it-and-how-to-get-started-52efd9cb52cf,* 2017.

[9] P. org, "PyTorch," *https://pytorch.org/,* 2019.

[10] A. MXNet, "PyTorch vs Apache MXNet," *MXNet documentation ,* 2019.

[11] A. MXNet, "Hybridize," *https://beta.mxnet.io/guide/packages/gluon/hybridize.html,* 2019.

[12] Q. W. P. X. X. C. Shaohuai Shi, "Performance Modelling and Evaluation of Deep Learning Frameworks on GPUs," 2018.

[13] J. L. W. D. a. D. L. Jie Liu, "Performance Analysis and Characterization of Training Deep Learning Models on Mobile Devices," 2019.

6. APPENDIX

The following appendices are provided to fulfil the requirements for COMP4560

6.1. Final Project Description

The project description is as follows :

"Benchmarking the Performance Analysis of MXNet for Image Classification on CPUs and GPUs"

At the conclusion of the project our goals were as follows:

- Discussing MXNet performance
- Providing appropriate Benchmarking measure
- Results discussion
- Detailed explanation on the performance analysis
- Discussion of advantages and disadvantages of the current deep learning frameworks

6.2. Study Contract

6.2.1. Project Title

Benchmarking the performance Analysis of MXNet for Image Classification on CPUs and GPUs

6.2.2. Learning Outcomes

An improved knowledge on deep learning frameworks, deep learning models and processors. To better understand the skills used for performance analysis.

6.2.3. Project Description

MXNet is a powerful open source deep learning framework which excels in programmability, portability and scalability. However, the most recent performance analysis for version 0.93 while the study will be conducting performance analysis for the latest version 1.5.1.
The study will compare MXNet's performance analysis with the latest version of Pytorch since both are direct competitors in the field of deep learning

6.2.4. Assessment

Report is a research project worth 90% of the final grade to be examined by Dr. Eric McCreath.
Presentation is worth 10% of the final grade to be examined by Prof. Weifa Lang.

6.3. Codes

The project uses the following codes :

Mourya Rishik, MXNet-vs-Pytorch-Benchmarks, "https://github.com/mouryarishik/MXNet-vs-Pytorch-Benchmarks",2019, last commit: 6795fb8.

The codes used in this repository are available on various online deep learning tutorials as mentioned by the developer.

The following codes perform the benchmark analysis for MXNet and Pytorch for the previous version. The study above works on the latest version for the Deep Learning Frameworks.

The study also uses *hybrid MXNet* instead of the standard MXNet model as follow :

```
#training model
mxnet.hybridize()
#loss function
#training model
#checking accuracy
```

Algorithm 7.1 Use of Hybridization for MXNet

The study shows improvement over the previous version and discusses other benchmarking metrics that can be used for performance analysis.

6.4. Test Runs

6.4.1. RESNET20

Test Run No.	MXNet	PyTorch
1.	29 s	1 min 33s
2.	28.6 s	1 min 32s
3.	28.7 s	1 min 34 s
4.	29 s	1 min 33s
5.	28.4 s	1 min 33s

Table 7.1 Five instances of training time for MXNet and PyTorch for RESNET20

6.4.2. VGGNET

Test Run No.	MXNet	PyTorch
1.	8.53 min	21.75 min
2.	8.49 min	21.31 min
3.	8.77 min	21.69 min
4.	8.47 min	21.82 min
5.	8.53 min	21.5 min

Table 7.2 Five instances of training time for MXNet and PyTorch for VGG