An Analytical Study of Major Deep Learning Framework & APIs

Department of Computer Science, California State University-Sacramento

Email Address

aviknayak@csus.edu

Abstract— **In the real world there are many Deep Learning frameworks in the AI space. Many companies use frameworks of their choice and various frameworks are implemented through different APIs. Our motivation is to do a comparative analysis both in terms of quality and quantity, so that companies using these frameworks and its Application Programming Interface(APIs) make an informed decision.**

Our goal is to show the difference in performance efficiency and ease of coding of the frameworks. The approach taken in this research endeavours is to run similar models of different APIs/Frameworks under standardized run environment (same dataset, hyper parameters etc.). This research is more intended towards highlighting the differences in terms of time complexity, ease of implementing the code and not towards accuracy directed performance.

Keywords— **Deep Learning, Keras, Eager execution, Estimator, Tensor flow, MXNET, MNIST**

I. Introduction

To understand the importance of the problem that we are trying to address is important to get the prospective of emergence of Deep Learning in today's day to day life of individuals. Right from recommendations in emails to understanding purchase behaviour of customers, Deep Learning has become very important to companies as it directly have an impact on their bottom lines. AI and Deep Learning is today's electricity.

Various companies use different frameworks in the market. Even though each framework has their positives, as well as their negatives, it is being observed recently that some frameworks and their APIs tend to outperform the others. Now this arises a question which framework/API is better? This motivated us to do analysis of various famous frameworks against one another. This is important as this will help companies get a cognizance of the best frameworks/APIs and directly implemented those that suits their needs.

Our approach in this research project is to consider three frameworks that are highly used in AI market. The frameworks that we have considered for analysis are: Tensor Flow (from Google), MXNET (from Amazon), Keras (Open Source). We also performed our analysis on Tensor flow APIs like Eager execution, Tensor flow Keras API, Tensor Flow Estimator API. In this process, we standardized the testing environment by considering the same for each:

1. Integrated Development Environment (IDE)
2. Programming Language
3. Dataset
4. Activation functions
5. Optimizer
6. Epochs
7. Model Structure
8. Hardware environment

Our contribution to this project involves carrying out various processes in order to have a meaningful comparison of the frameworks/APIs. We did this through following various steps:

1. Research various related papers [1]
2. Created the test environment which involves downloading various dependencies that are critical for the success of the project.
3. Downloading and Pre-processing the Dataset(MNIST-Hand written digits and Cats and Dogs from Kaggle)
4. Implementing each of the Deep Learning programs using the concerned framework/API.
5. Pictorial representation of the each framework results and their comparison on each parameter.
6. Analysing quantitatively as well as qualitatively the performance and ease of usability of code.

The paper is organized starting by an introduction on to the overall project and the frameworks involved and looking at each framework and API concerned individually and briefly describing each and the methodology use. After we will look at the performance of individual framework and will show the comparative analysis on different parameters.
Finally, the paper concludes showing the individual contribution in this research project, drawing the final conclusion and ending with the acknowledgement and citation.

II. Problem Formulation

Our project is focused on processing in recognizing hand-written digits of MNIST (Modified National Institute of Standard and Technology) dataset using Convolutional Neural Network at its core. In this process we observe the time taken by each framework/API and the number of lines of code to do the same task of Image Recognition, observing the complexity of the implemented code.

The most important part of the challenge in dealing with this

be standardised, various library dependencies must be taken care when implementing various models and there may be conflicts.

The input to the program would be the dataset images in form of multi-dimensional matrix split into training and testing set and the output would be the accuracy of the model in correctly classifying the images based on its learning.

III. Deep Learning Frameworks/APIs

There are many deep learning frameworks that have been advocated by their respective maker. Some deep learning frame works like Keras, Tensor Flow, MXNET, CNTK have been going strong due to community support or massive research-based investment by deep pocketed companies. We have taken three major deep learning frameworks. Also, we have considered the APIs of one prominent deep learning framework used in the market and analysed their comparative performance. The frameworks considered by us for comparison in this analysis are Tensor Flow, MXNET, Keras . These are the top Deep Learning frameworks in the market [2]. Many big companies use these frameworks to drive their technology-based products. We would be briefly describing these frameworks.

A. TensorFlow

TensorFlow was released by Google in November 9 2015[3]. It was the evolution of a machine learning system developed by Google earlier called DistBelief [3]. TensorFlow is very robust open source software AI library which is being used to develop powerful neural network. It is capable of running in Central Processing Unit (CPU) and Graphic Processing Unit (GPU) based systems. The library is used both in lots of academic and non-academic research and in production systems. TensorFlow can be coded in programming languages like C/C++, Python, Java, Go. This library can be installed in both Windows as well as Linux based operating system. TensorFlow uses various high-level APIs that can be used for the execution of the library although we can use TensorFlow without using the APIs. The major high-level APIs used by the TensorFlow are 1. Keras 2. Eager Execution 3.Estimator. There are many APIs version from r1.0 to r1.12. The figure below shows macro view of the APIs and overall landscape. (Fig1)

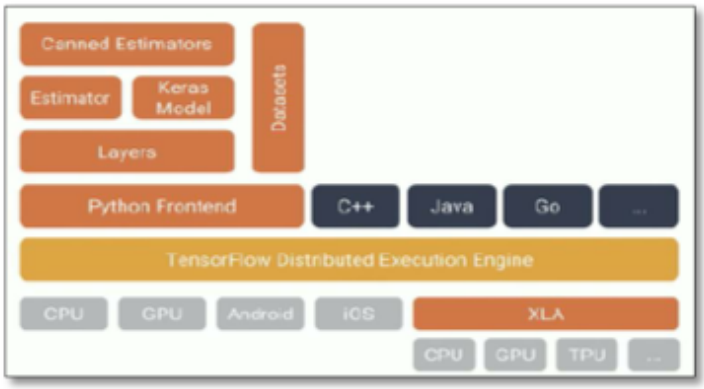

Fig. 1 A Snapshot of the TensorFlow expanding Architecture (Source: https://www.altoros.com/blog/a-broad-spectrum-of-TensorFlow-apis-inside-and-outside-the-project/)

1) Keras API: Keras API is an high level API and sits on top of the python frontend and has very easy to use interface. It is user friendly and designing and building the neural network is easy. The Keras API provides great advantage in being modular and comparable. There are not many restrictions when using Keras since it provides building blocks to design and build large or small neural network models. The APIs are easier to extend if need be and express new ideas for research or in real work implementation [3].

2) *Eager Execution API:* This is a high level TensorFlow API. This API is primarily focused in fast execution of the model without building graph. This API is majorly used for research and experimentation which require quick testing and result generation. This API is handy because it doesn't wait for the generation of the graph and operations run concrete values directly. This is intuitive API and doesn't require graph control and can uses basic python coding and data structure. As this API is close to python data structure, debugging is easy and any tool used for python debugging can be used. This API reduces boiler plate as well [4].

3) *Estimator API:* This API is specifically created by Google to handle large case data. Google provides both customizes Estimator solutions and also provides scope of building custom models. Estimators are very useful when running on distributed systems and can run on CPU and GPU without recording the model. Estimator is very useful in the distributed system. The best part of estimator is that it builds expressive graphs for the user automatically and provides a safe distributed training loop. "When writing an application with Estimators, you must separate the data input pipeline from the model. This separation simplifies experiments with different data sets" [5].

B. Keras

The framework Keras is a very famous neural network framework and is supported by opensource community, although its main author is François Chollet [6]. It can work with TensorFlow or CNTK of Microsoft. It is highly user friendly and provides high level of abstraction to the developers which helps in implementing the model faster and quickly checking the results. It is widely used in AI community for its ease of use. Although it is found to be not as robust or scalable as other major framework/library like TensorFlow or CNTK, but still provides library for developers getting induced to Deep Learning library. Keras for from the onset was considered as an interface than a standalone machine learning library thus it acts and an excellent API [6].

C. MXNET

The framework MXNET that is supported by Amazon is one of the famous Deep Learning libraries. It was developed by Apache software foundation [7]. MXNET has an API called gluon. This library was chosen by Amazon for its AWS cloud service driver, although other frameworks can also be used in the cloud. The biggest advantage of MXNET Deep Learning library is that it can be scaled to work on massive amount of data and can work on massive distributed architecture. This framework supports multiple languages like MATLAB, Python, R, Scala, Julia, Perl and this frame work can work on CPU and GPU too including multiple GPUs.

1) Gluon API: Gluon is an API provided for MXNET framework and supports MXNET framework. It provides and

easy to use interface and it can work on both CPU driven machines as well as GPU driven machine. It doesn't used Numpy array rather "nd" array is used for data manipulation. However there are libraries to convert Numpy array to "nd" array. Nd array is provided by MXNET.

A macro level version snapshot of the frameworks under analysis is shown below.

TABLE I
FRAMEWORKS USED FOR THIS COMPARATIVE STUDY

Sl no	Major Frameworks		
	Framework Name	Newest Version	Licence
1	TensorFlow	1.12	Apache License 2.0
2	Keras	2.2.0	MIT
3	MXNET	1.3.1	Apache License 2.0

IV. SYSTEM DESIGN AND ALGORITHM DESCRIPTION

The analysis of the project wherein we compare the performance and ease of code usability required us to maintain a standardised environment which not only included standardised runtime background hardware. It has same dataset and hypermeters, also in our analysis, we used Google Collaborator as the testing environment, this helped to standardise the IDE and the hardware quickly and took care of many dependencies. We even went to the level of standardising the model and its hyperparameter. It was of critical importance to standardise the overall environment as any changes in the environment would have biased the results to favour one framework over another. The further details of the individual standard parameters and algorithm are given future below.

A. The Integrated Development Environment

The Integrated Development environment used was Google Collaborator. The IDE is a free development environment provided by Google. The choice of this development environment was due to the following reasons

1. Free to use.
2. Choice of CPU, GPU.
3. Minimise library dependencies issues.

The hardware specification of the IDE is given below

1) *Disk Information:* The specification of the disk of the IDE is given in (Fig 2).

```
Filesystem      Size  Used Avail Use% Mounted on
overlay          40G  7.0G   31G  19% /
tmpfs           6.4G     0  6.4G   0% /dev
tmpfs           6.4G     0  6.4G   0% /sys/fs/cgroup
tmpfs           6.4G  4.0K  6.4G   1% /var/colab
/dev/sda1        46G  8.5G   37G  19% /etc/hosts
shm             6.0G     0  6.0G   0% /dev/shm
tmpfs           6.4G     0  6.4G   0% /sys/firmware
```

Fig. 2. A Snapshot of the Disk Specification

2) *CPU/GPU Information:* The processor used to run Collaboratory was from Intel although, Google also uses GPU from Nvidia processors. The Specification of the CPU processor is given below in Fig 3.

```
vendor_id       : GenuineIntel
cpu family      : 6
model           : 63
model name      : Intel(R) Xeon(R) CPU @ 2.30GHz
stepping        : 0
microcode       : 0x1
cpu MHz         : 2300.000
cache size      : 46080 KB
```

Fig. 3. A Snapshot of the CPU Specification

Overall the IDE hardware specifications are as given below:

1. GPU: 1xTesla K80, having 2496 CUDA cores, compute 3.7, 12GB (11.439GB Usable) GDDR5 VRAM.
2. CPU: 1xsingle core hyper threaded i.e(1 core, 2 threads) Xeon Processors @2.3Ghz (No Turbo Boost), 45MB Cache
3. RAM: ~12.6 GB Available.
4. Disk: ~33 GB Available.

B. Dataset

The dataset we used is a well-known dataset. This dataset is called the MNIST (Modified National Institute of Standards and Technology database) dataset. It contains the handwritten digits images from 0 to 9 in various shapes and style. A snapshot of the images in the dataset in given in the below Fig 4.

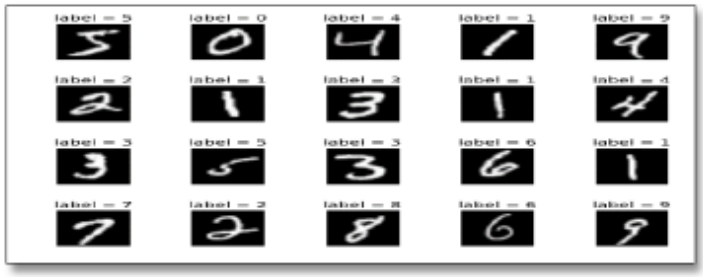

Fig. 4. A Snapshot of the MNIST Dataset that were used

The dataset had 55,000 training records each of 784 columns as features and 10,000 testing/validation records with similar features columns count. There were 10 label columns that was indicating 0 to 9.

C. Hyperparameters

As much as possible we have tried to standardise the hyperparameter involved in construction and execution of the Convolutional Neural Network model in all the frameworks. The Hyperparameters used are given below which were common in every framework.

1) Training Parameters
Learning Rate = 0.001
Number of Steps = 20
Batch Size = 128
2) Network Parameters
Number of input = 784 # MNIST data input (shape: 28*28)
Number of Classes = 10 # MNIST total classes (0-9 digits)
Dropout = 0.25 # Dropout, probability to drop
Activation Function= Relu
Optimisation Function= Adam

D. Standard Algorithm Description (Model Used)

The model used was a Convolutional Neural Network (CNN). CNN is a type of deep neural network that works best with image processing, it follows the way visual images are processed by visual cortex of living organism. When compared to other neural network CNN models were found to provide much better image feature extraction capability and is thus most commonly used for visual processing. The model has Convolution layer, Max Pooling layer, Flatten layer, Dense layer, Drop out Layer and Activation function too. The snapshot of the model that was mostly used while implementing the individual frameworks is given below in Fig 5. This model standardisation was necessary as different model may give different results.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 24, 24, 32)	832
max_pooling2d (MaxPooling2D)	(None, 12, 12, 32)	0
conv2d_1 (Conv2D)	(None, 10, 10, 64)	18496
max_pooling2d_1 (MaxPooling2	(None, 5, 5, 64)	0
flatten (Flatten)	(None, 1600)	0
dense (Dense)	(None, 1024)	1639424
dropout (Dropout)	(None, 1024)	0
dense_1 (Dense)	(None, 10)	10250

Fig. 5. A Snapshot of the CNN Model that was used

The CNN network uses three major types of layers. The first layer is the convolution layer which helps in feature extraction, second layer is the pooling layer which is majorly used to reduce the dimensionality of the input multi-dimensional data. The last layer is the activation layer which helps in incorporating non-linearity in the data. It is to be noted that CNN is a feed forward network and the output of the previous layer. The convolution layer does most of the computationally intensive work that is intended. We also use a concept of back propagation and gradient decent to optimise the waits and biases in the model system so that it increases in accuracy to detect the image accurately. There may be a dropout layer added that helps in reducing the overfitting of the model.

E. Programming Language Used

The language that was used was used was Python 3.7. Python being one of the most versatile language was preferred over any other language

V. Experimental Evaluation

We now are going to describe the entire process starting from the methodology to the results of the experiment along with the observation and conclusion based on it.

A. Methodology

We start by describing the dataset. The dataset used was MNIST database. MNIST stands for Modified National Institute of Standard and Technology. It has many different type of datasets but we used Handwritten Digits dataset. This dataset consists of 65,000 images which includes 55,000 training set and 10,000 testing set.

To talk a bit more about the experimental setting, we made sure that the Deep Learning frameworks/APIs are tested under standardised test conditions. We tried to maintain the standardisation in-spite of various dependencies issues that persist when dealing with different eclectic mix of Deep learning libraries. We even used the same model structure and hardware/software environment across all the Deep Learning frameworks.

The metrics used to compare the Deep Learning Models are listed below.

1. Time Taken
2. Ease of Coding

Note since Ease of Coding is intuitive, we have assigned a scale to map the coding friendliness with 0- Least Friendly to code, 1-Low, 2 – Normal, 3-Easiest to code.

The method which was implemented was the time function that noted the time of the model run before and after which was found different. There is python function from standard library that was used to implement this method. The result from this method showed us the time elapsed during the model run and this time was compared across frameworks. The ease of coding was visually and intuitively compared looking at the number of lines of codes taken to implement the same functionality and complexity of the code.

B. Results & Individual Conclusive Observation.

In our analysis the results were observed after looking at various random run.We did many runs and found the results(time taken) to be farely around the same range and consistant.There was also remarkable differences in coding structure across various frameworks and the API. The differnce not only included the syntax but also number of lines of code to implement the same model. We will showcase the result below.

1) TensorFlow + Keras API

The API Keras is one of the API of TensorFlow and makes coding easy and quick to understand and implement. We implmented the code to do image recongnition using CNN network with a model shown in Fig 5 .

1-A) Observations & Results:

As can be seen in Fig 6, the time taken by the model to run is 1500.74 seconds when running in CPU mode but when running under GPU, the model ran in 111.74 seconds.Thus we see than CPU takes almost 13 times higher than running in GPU.

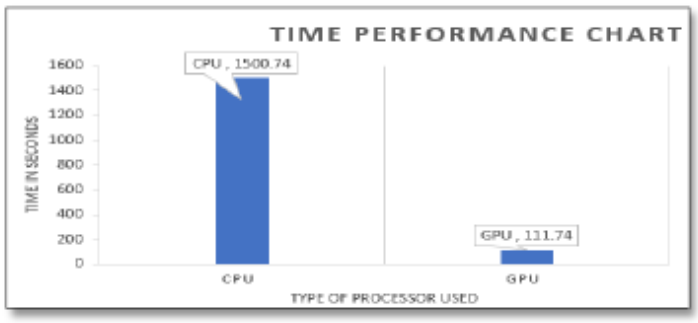

Fig. 6. TensorFlow with Keras API Time Performance

Coding in TensorFlow using Keras API is very easy and we observed a high level of abstraction in the coding .11 lines of codes were used to build and compile the standard CNN model which is quite nice and thus we give a scale of 3 to this API.

2) *TensorFlow + Estimator API*

The Estimator API is also another API which TensorFlow uses to encapsulate training, evaluation, export serving and prediction. It is like before we used this estimator to evaluate the accuracy of image recognition using our standardised CNN model

2-A) Observations & Results:

As can be seen in the Fig 7 we found out that when we run the model using estimator the time required to run the CNN model in Colab using CPU was 9.78 sec which was way less than using Keras API.The time taken by the model to complete the entire process when running in GPU mode was 6.90 seconds.

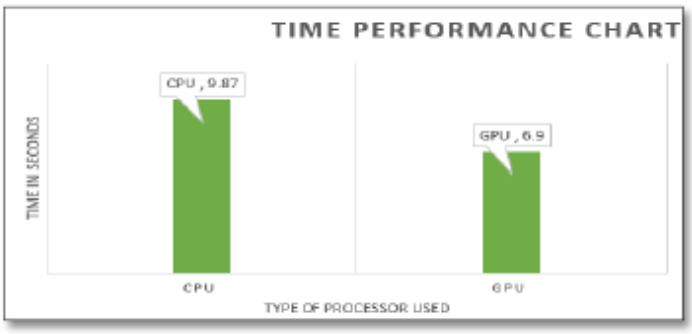

Fig. 7. TensorFlow with Estimator API Time Performance

Code complexity is higher in Estimator API than Keras API. We see that to implement the same model in estimator we have used 32 lines of code which was much higher than Keras API. This is quite high amount of lines of code, so we give it a scale of 1 that is low code friendly.

3) *TensorFlow + Eager Execution API*

The eager execution API is quicker then Keras API. This is because when we use Eager Execution API the computational graph is not build.

3-A) Observations & Results:

As we can see from the Fig 8 we found that the Eager Estimator API is very fast, and the model run in 13.57 seconds in CPU and in GPU the standardised CNN model run in 5.98 seconds which is by far the fastest so far. We can also intuitively see that this kind of fast executing API is quite useful when we look at task where quick debugging is needed.

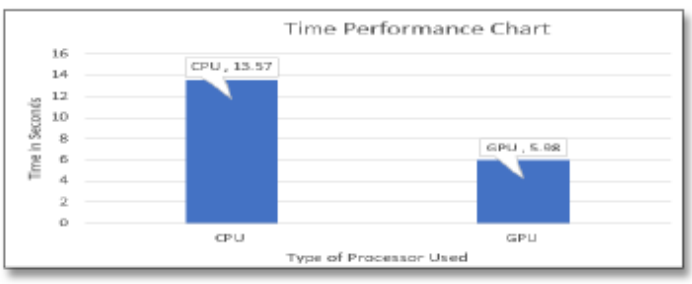

Fig. 8. TensorFlow with Eager Execution API Time Performance

The Coding complexity and user friendliness is a bit high, it mostly uses Python kind of structure implementation for building the various component of the code. Although it's not as easy to code as Keras API, we give it a scale of 2 which is Normal in coding as it uses python Classes to implement which is understandable by regular coders. The number of lines of primary codes used by this API to implement the same standardised model is 27.

4) *MXNET + Gluon API*

MXNET the Amazon marketed framework works with Gluon API which makes it easy to implement and is intuitive to learn and execute. We have used the same standardised CNN model that we have used for the rest of the APIs in TensorFlow.

4-A) Observations & Results:

The time required by the CNN model implemented with Gluon API is quite Page numbers, headers and footers must not be used. The time required for the model to be executed by CPU was 241.01 Seconds and the time it ran in GPU was 256.52 seconds as can be seen in Fig 9.

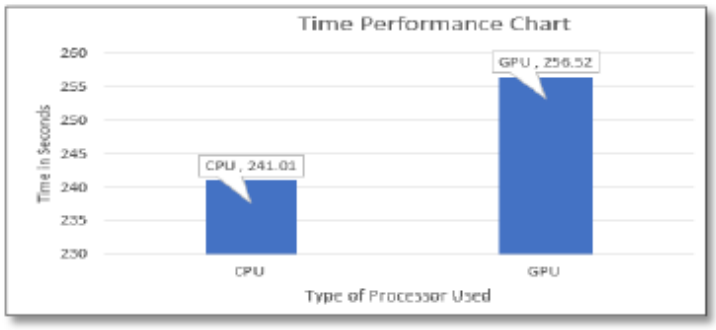

Fig. 9. MXNET with Gluon API Time Performance

The coding is not much difficult or verbose and is easy to understand,to implement the same model it took 26 total lines of codes which included the compilation and running the model , it still seems to be bit more code than Keras and thus we have assigned it a scale of 2 which is Normal.

5) *KERAS*

Keras is a deep learning frame work that can also be used independently and need not sit on top of any other framework as API. We tested Keras for its performance too using the CNN model that we have used for other tests and under the said standardised control environment.

5-A) Observations & Results:

We observed that on a stand-alone basis Keras performed very badly in CPU mode and it took a massive 1147.14 seconds and in GPU mode it too 256.52 seconds. This is much slower when it is used as an API with TensorFlow. The performance between CPU and GPU can be seen in the Fig 10. Keras has been the slowest when compared to the other Framework/API that we have analysed.

We can safely assume that the probability of implementation of Keras in production application where fast processing is very much needed is a rare probability.

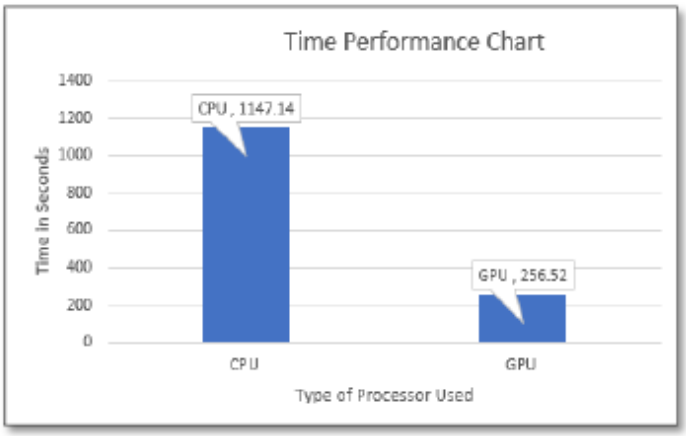

Fig. 10. Framework KERAS Time Performance

The bright side we see that the code is very easy, short and only takes 11 lines of code to build the equivalent model which is excellent when compared to other implementation this we assign it a scale of 3.

6) *All Frameworks Performance Comparison*

All the frameworks and the APIs that were analysed showed considerable varied performance like eager execution while being very fast when compared to Keras was not as easy to code as Keras which in turn was much slower to execute and implement.

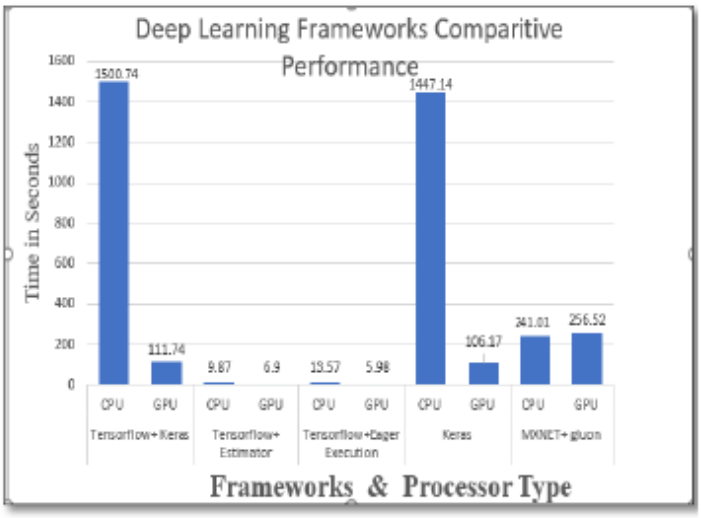

Fig. 11. Comparative Time Analysis Chart of Frameworks/API.

As we can see from the above chart Keras took the most time and TensorFlow Eager Execution took the least time when running in a GPU. From this we can conclude that in production systems and in industry application Tensorflow Estimator is much better option than Keras which is much slower also if computational graph is not needed the Tensorflow eager execution can be used too.

If we see in Fig 12, we see that Keras is the framework/API that has the highest level of abstraction and is easiest to code. The code of Keras are easy to understand and implement. It has many build-in functions that helps us to keep away with using iteration writing many function calls and implement optimisation and activation layers very easily, MXNET Gluon API comes close to the easiness of implementation as Keras but other framework/API require more code to implement the same functionality or model.

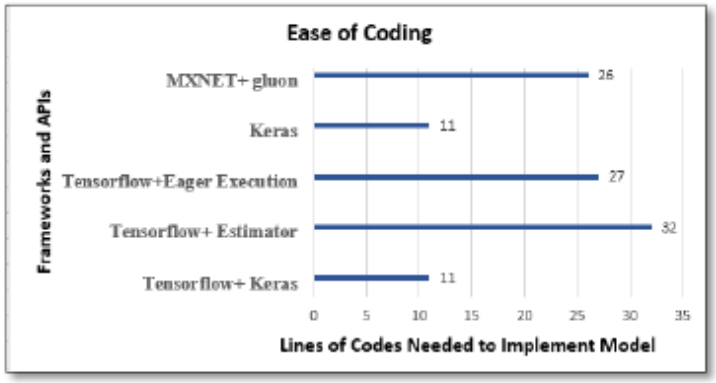

Fig. 12. Comparative Code Ease of Use Analysis Chart of Frameworks/API.

VI. DL FRAMEWORKS/APIS LITERATURE SURVEY

There has been some research work conducted in this field of study although most of the research has been to compare some of the famous deep learning frame works as such and not their individual APIs. In our analysis we have additionally compared the three major APIs namely TensorFlow eager execution, TensorFlow, Keras, TensorFlow Estimator.

Some research paper that have done comparative study. A paper "A Comparative Study of Open Source Deep Learning Frameworks " by *Ali Shatnawi, Ghadeer Al-Bdour, Raffi Al-Qurran and Mahmoud Al-Ayyoub*[1], has tried to compare deep learning frameworks like TensorFlow, Theano and CNTK using CIFAR-10 dataset and found that CNTK was superior when compared to the rest in most of the test run .The metric used were the runtime, memory consumption, CPU and GPU utilization. They did many run in different processing environment and found out that the CNTK was performing better most of the time against the other frameworks taking the above discussed metrics into consideration.

Another study was done through the research paper "Comparative Study of Deep Learning Software Frameworks" by *Soheil Bahrampour, Naveen Ramakrishnan, Lukas Schott, Mohak Shah* [8]. In this paper Caffe, Neon, TensorFlow, Theano, and Torch framework were compared, and the comparison was done on three aspects namely speed, extensibility and hardware utilization. In this endeavour performance both in CPU based and GPU based machines were observed. In this analysis it was noticed that Theano and Torch were the most easily extensible frameworks among all compared. It was also observed that Torch was better suited deep architecture on CPU than Theano. The paper also highlighted that Caffe was better suited for evaluating performance of deep architectures.

Our research is different from the above research from the fact that we endeavoured to analyse how different APIs of the same Deep Learning Framework give different result and we also compared APIs of TensorFlow with APIs of MXNET namely Gluon. We also unlike the other used openly available free resources and IDE like google Collaboratory to implement our model, this helped us to bring to our disposal powerful machine which gave accurate results to our project work compared to the rest who mostly used their own personal computers and laptops.

VII. Conclusions

Our research and analysis were thorough and repetitive; we used top of the line open source available infrastructure to carry out our research. From our analysis we found that different frameworks/APIs are different in their performances. While TensorFlow Eager Execution API was found to be faster, Keras API was found to be the easiest to code and implement. We draw an intuitive understanding that no APIs or Framework is neither good nor bad rather different situation demands implementation of different APIs and framework. In a situation which requires fast execution of code like production system, a faster Framework/API is needed like TensorFlow Eager Execution which is based on academic setting of learning a framework/API like Keras.

In the analysis we also found that MXNET may be the frameworks of choice in Amazon Web Services(AWS)[9],it performed rather poorly in an environment created by Google(Google Collaboratory).The best runtime we received was from TensorFlow Eager Execution in GPU setting, the runtime was 5.98 seconds and Keras just took 11 lines of code to implement a multilayer CNN model which was very less when compared to others.

VIII. WORK DISTRIBUTION

The project was done by two Graduate Students namely Avik Nayak and Anshul Jain. The work was divided among both so that the project can be completed in time. The work distribution is listed below.

1) Avik Nayak
   1. Went through research paper and related topic.
   2. Implemented two frameworks/API Programs.
   3. Standardized the Environment.
   4. Wrote the Report.
   5. Comprehensively review presentation and made changes as was necessary.
   6. Learned New Frameworks/API

2) Anshul Jain
   1. Went through research paper and related topic.
   2. Implemented Three Framework/API Programs.
   3. Standardized the environment.
   4. Comprehensively Reviewed the report and changed.
   5. Prepared Presentation.
   6. Learned New Frameworks/API

IX. Learning Experience

The project gave us a lot of scope to learn new information and gain new knowledge. Through this project we could get a deeper insight into the comparative performance of various famous frameworks/APIs. This lead us to better appreciate the fact that one framework cannot be used in different situations. Some frameworks are easier to learn and code and are suitable in academics setting, while some are good for production environment and some are good for testing and debugging. We also learned to implement new frameworks and got deeper understanding of major TensorFlow APIs.

This project helped us understand open source IDE that is Google Collaboratory better and how to utilize that tool to collaborate and learn and implement very focused AI, Deep Learning programs. Writing the project report also exposed us to the area of professional research and taught us what level of academics' standards to maintain while writing reports. This will go a long way in helping us in our future projects that will involve writing research papers.

Acknowledgment

We would like to give our heartful thanks to our Professor Haiquan Chen without whose constant support and guidance this project would not have been completed on time. We also would like to thank the authors of all the documents and papers who have provided us with material that helped us gain more knowledge in the topic.

References

[1] Ali Mohammd ShatnawiAli Mohammd ShatnawiMahmoud Al-AyyoubMahmoud Al-Ayyoubghadeer albdourRaffi Louy Al-QurranRaffi Louy Al-Qurran "*A Comparative Study of Open Source Deep Learning Frameworks*". International Conference on Information and Communication Systems At: Jordan (30 Apr 2018),p. 1-5.

[2] (2018) Top 8 Deep Learning Frameworks [Online]. Available: https://www.marutitech.com/top-8-deep-learning-frameworks/

[3] (2018) The TensorFlow Keras API Website [Online]. Available: https://www.TensorFlow.org/guide/Keras

[4] (2018) The TensorFlow Eager Execution API Website [Online]. Available: https://www.TensorFlow.org/guide/eager

[5] (2018) The TensorFlow Estimator API Website [Online]. Available https://www.TensorFlow.org/guide/estimators

[6] (2018) The Keras article Wikipedia Website [Online]. Available https://en.wikipedia.org/wiki/Keras#cite_note-2

[7] (2018) The MXNET article Wikipedia Website [Online]. Available https://en.wikipedia.org/wiki/Apache_MXNet

[8] Soheil Bahrampour, Naveen Ramakrishnan, Lukas Schott, Mohak Shah" *Comparative Study of Caffe, Neon, Theano, and Torch for Deep Learning*". Bosch Research and Technology Center At: North America (30 Mar 2016), *p 1-5.*

[9] (2018) MXNet - Deep Learning Framework of Choice at AWS [Online]. Available https://www.allthingsdistributed.com/2016/11/mxnet-default-framework-deep-learning-aws.html