PyTorch vs TensorFlow and Other Insights: A Comparative Study on Deep Learning Frameworks

Suyash Patel, Aditya Kumar Verma, LakshmiVihari Mareedu
Arizona State University
Tempe, USA

***Abstract*—In this study, we compare the training and inference times using PyTorch and TensorFlow, two well-known deep learning frameworks in centralized and distributed settings. To provide a thorough comparison, we run tests with a range of models, batch sizes, and input shapes. Our findings demonstrate that there is no clear winner among the frameworks and that performance depends on the particular model architecture and task. Additionally, we investigate how batch size affects inference time and examine distributed training methods like PyTorch's DistributedDataParallel and DataParallel. Researchers and practitioners using these frameworks can benefit from our findings.**

I. Introduction

With numerous uses in many fields, including speech recognition, natural language processing, and computer vision, deep learning is gaining popularity. TensorFlow and PyTorch are two of the most well-liked deep learning frameworks. The performance of each framework varies depending on the model architecture, batch size, and other factors, making it difficult to select the best one for a given task. Detailed comparisons of PyTorch and TensorFlow are provided in this document, along with information on distributed training methods, ideal inference batch sizes, and other topics.

II. Problem Definition and Motivation

The main goal of our study is to evaluate PyTorch and TensorFlow's training and inference times as well as their suitability for distributed training. There are many deep learning frameworks available today, and choosing the right one can be a challenging task. Among the most popular are PyTorch and TensorFlow. Both frameworks have their strengths and weaknesses, and choosing between them depends on the task at hand, the available resources, and the preferences of the developer.

One way to compare the two frameworks is to evaluate their performance on a specific task, such as image classification. This can be done by using pre-trained models and comparing their accuracy and resource consumption. By doing so, we can gain insights into the strengths and weaknesses of each framework, and identify which one is better suited for a specific task. This comparison is intriguing because it can guide professionals and researchers in selecting the best deep learning framework for their projects. We also explore factors to be taken into account when using framework-native APIs like PyTorch's DistributedDataParallel (DDP) and DataParallel for distributed training. Additionally, we also aim to understand the relationship between inference time and batch size.

III. Background and Literature Review

To provide context and background for our study, we examined several articles and research papers that focus on the performance comparison of PyTorch and TensorFlow, distributed training strategies, and batch size optimization. A key reference that greatly influenced our project is:

- Canziani, A., Paszke, A., & Culurciello, E. (2016). An Analysis of Deep Neural Network Models for Practical Applications. arXiv:1605.07678.

This influential paper analyzes the trade-offs between accuracy, memory footprint, and computational requirements of various deep learning models. Our project not only validates and reinforces their findings but also expands on their research by conducting a comparative analysis of popular deep learning frameworks such as PyTorch and TensorFlow.

Building on the insights from Canziani et al.'s paper, our project further delves into the practical aspects of implementing deep learning models, focusing on DataParallel and DistributedDataParallel parallelization techniques, as well as the effects of varying batch sizes on inference times.

In addition to Canziani et al.'s paper, other important references that informed our research include:

- Paszke et al. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. Advances in Neural Information Processing Systems.
- Abadi et al. (2016). TensorFlow: A System for Large-Scale Machine Learning. OSDI.
- Sergeev and Del Balso (2018). Horovod: Fast and Easy Distributed Deep Learning in TensorFlow. arXiv:1802.05799.

These references, along with Canziani et al.'s paper, provided a solid foundation for our study, allowing us to contribute valuable insights into the practical implementation of deep learning models using popular frameworks such as PyTorch and TensorFlow.

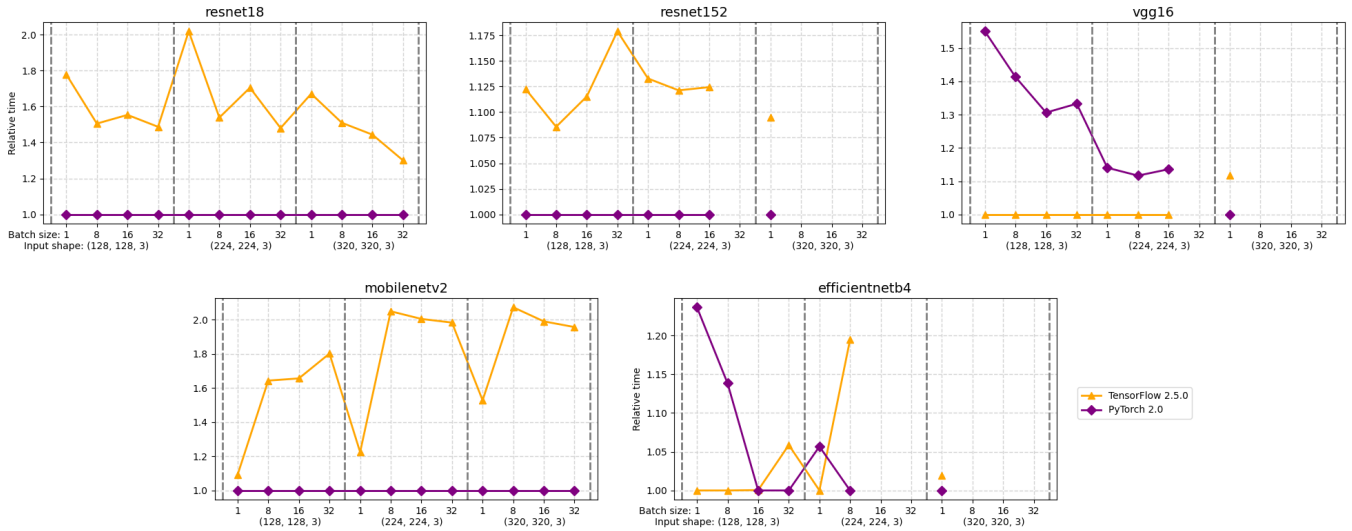

Fig. 1: Benchmarking relative training times for various models with different hyper-parameters

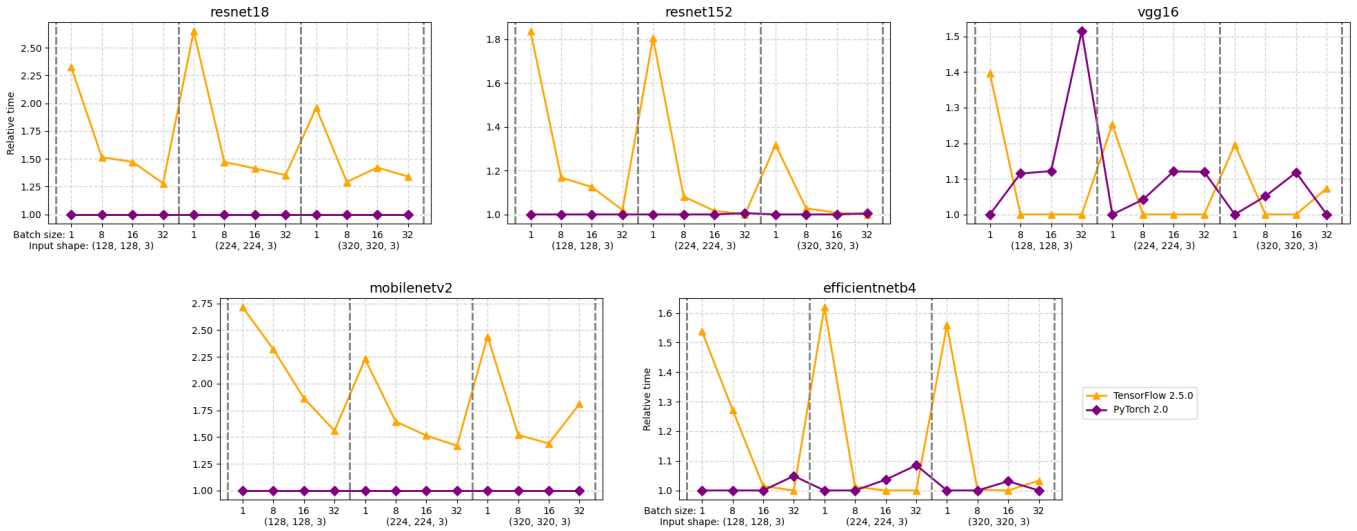

Fig. 2: Benchmarking relative inference times for various models with different hyper-parameters on a single GPU

IV. Proposed Methods and Algorithms

By evaluating PyTorch and TensorFlow's capabilities in training and inference tasks, our study seeks to benchmark these two frameworks. The methods and algorithms listed below were used to accomplish this:

A. Dummy Dataset Generation

We chose to use a dummy dataset to avoid the need for intricate data transformations when changing batch sizes and input shapes. The dataset consists of tensors that were generated at random and have defined batch sizes, input shapes, and channel counts. With this strategy, we were able to completely ignore the impact of loading times and data preprocessing and concentrate only on the frameworks' performance.

B. Model Selection

We chose several standard deep learning models to ensure that our performance comparison covers a wide range of architectures. These models include:

- ResNet18 and ResNet152: Residual Networks with 18 and 152 layers, respectively, known for their excellent performance on image classification tasks.
- VGG16: A 16-layer deep convolutional neural network that demonstrated strong performance on the ImageNet dataset.
- MobileNetV2: A lightweight deep learning model designed for mobile and edge computing applications, with an emphasis on low-latency and low-power consumption.
- EfficientNetB4: A scalable neural network architec-

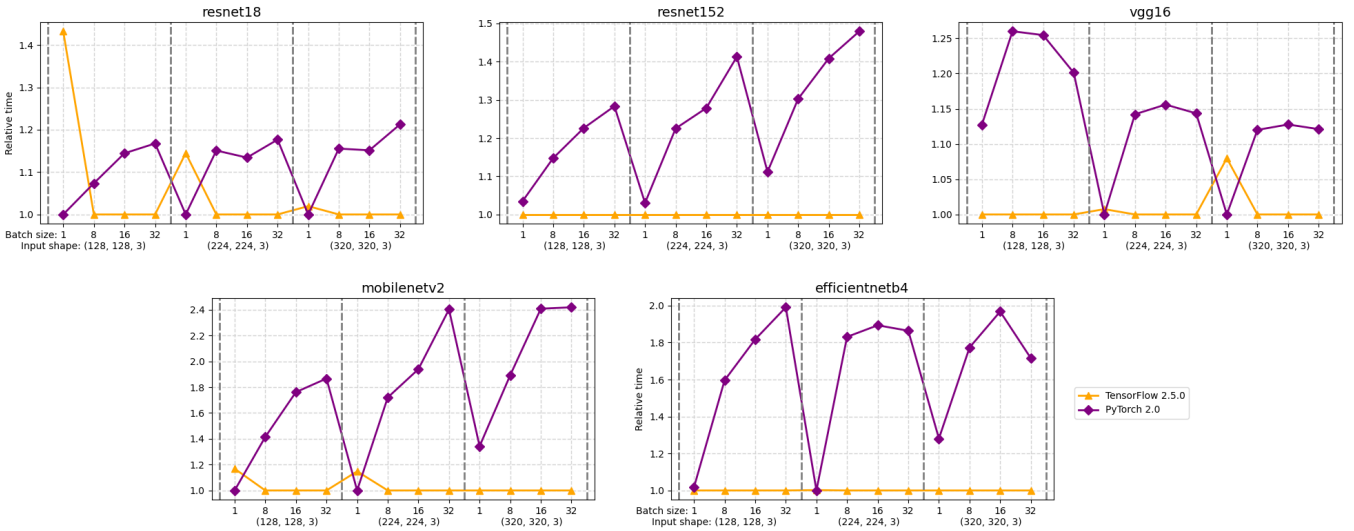

Fig. 3: Benchmarking relative inference times for various models with different hyper-parameters on a single CPU

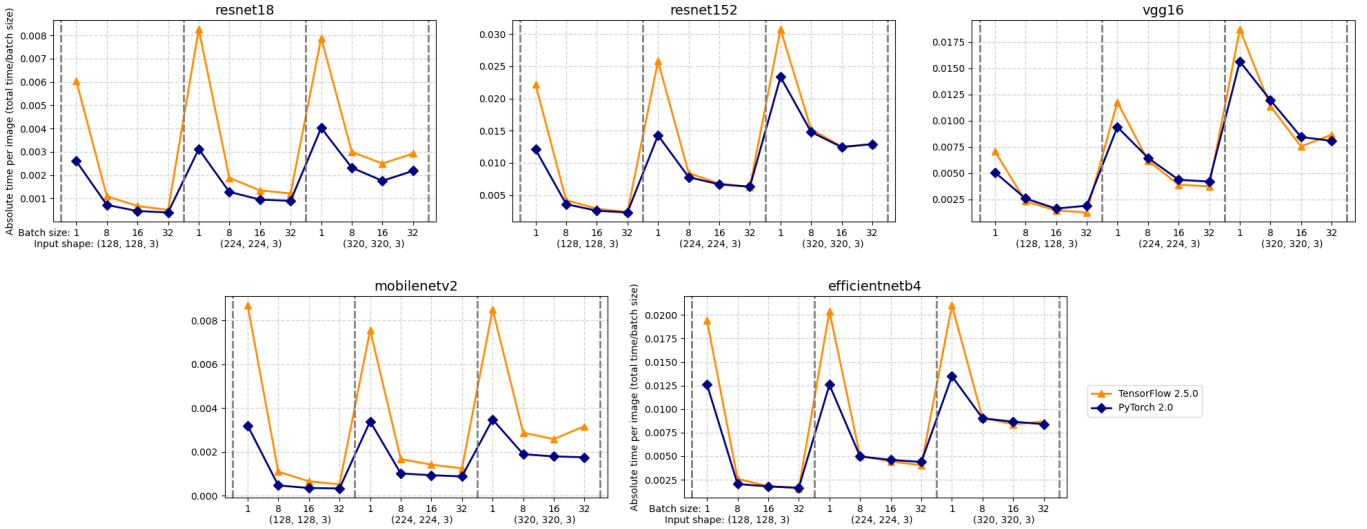

Fig. 4: Relationship between inference and batch size

ture that strikes a balance between model size and performance across a variety of computational budgets.

C. Experimental Design

We experimented with different batch sizes and input shapes to evaluate PyTorch and TensorFlow's performance. Using 100 runs for each configuration, we calculated the average training and inference times. The following queries were the focus of our experiments:

- How is the length of training and inference time impacted by the deep learning framework selected?
- How does the performance of each framework change as a result of batch sizes and input shape variations?
- Do any particular models or tasks consistently perform better than the other for either framework?

D. Distributed Training Techniques

We compared DataParallel and DistributedDataParallel's performance for distributed training in PyTorch to further examine the effectiveness of each framework. For large-scale deep learning applications, these techniques are essential because they enable parallelism across multiple GPUs. Our tests aimed to evaluate the effectiveness and scalability of each strategy and to spot any potential bottlenecks or constraints.

Our study provides a thorough comparison across different model architectures, batch sizes, and input shapes by utilizing these techniques and algorithms. Users will

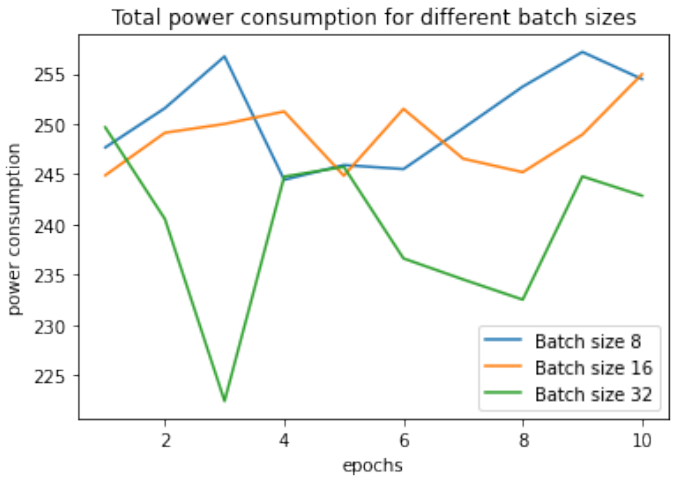

Fig. 5: Relationship between power consumption and batch size

be guided by the results in choosing the best deep learning framework for their unique needs and computational capabilities.

V. Experimental Environments and Setup

We utilized ASU's Agave Cluster for our experiments, which provided us with two distinct hardware configurations for single GPU and distributed training scenarios.

1) Single GPU Configuration: For single GPU experiments, our hardware setup consisted of 4 GB of memory, one NVIDIA GeForce RTX 2080 Ti GPU, and one CPU core. We chose this reasonable configuration to ensure high resource availability during the execution of our experiments, as we had multiple models to test with varying batch sizes and input shapes. By opting for this setup, we avoided potential bottlenecks that might arise from resource constraints.

2) Distributed Training Configuration: For distributed training experiments, we used a more robust hardware configuration within ASU's Agave Cluster. This setup consisted of a single node with 8 GB of memory, two NVIDIA GeForce RTX 2080 Ti GPUs, and two CPU cores. This configuration allowed us to effectively compare the performance of DataParallel and DistributedDataParallel in PyTorch.

By employing these hardware configurations, we ensured that our experiments accurately reflected the performance in both single GPU and distributed training scenarios. For benchmarking, we performed the experiments 100 times to obtain the mean training and inference time plots, providing a reliable basis for comparison between the frameworks.

VI. Evaluation Results

Our evaluation results can be summarized as follows:

A. Comparison of Training Times

We discovered that, in terms of training time, there is no clear winner between PyTorch and TensorFlow because the performance is dependent on the particular model, task, and batch size. For example, PyTorch outperformed TensorFlow when running the ResNet18, ResNet152, and MobileNetV2 models, while TensorFlow triumphed when running the VGG16 and EfficientNetB4 models. Figure 1 shows the relative training times on PyTorch and TensorFlow for various models with different hyper-parameters. Note that the non-existence of some data points implies that the GPU memory wasn't enough to accommodate the tensors and therefore, the training failed. As a consequence, the training times weren't obtained.

B. GPU and CPU Inference Time Comparison

With the exception of VGG16, PyTorch outperformed all of the tested models for GPU inference. Figure 2 shows the relative inference times on PyTorch and TensorFlow for various models with different hyper-parameters on a single GPU. Contrarily, with a few notable exceptions, TensorFlow consistently outperformed CPU inference. TensorFlow's static computation graph, which enables more aggressive optimizations, and use of the XLA compiler for additional performance enhancements can be credited for this. Figure 3 shows the relative inference times on PyTorch and TensorFlow for various models with different hyper-parameters on a single CPU.

C. Relationship Between Inference Time and Batch Size

It is evident from Figure 4 that as the batch size increases, the inference time decreases up to a limit. There is a clear trend that can be seen across models and this can be verified statistically as well using hypothesis testing. This observation is critical for serving models on edge devices, where determining the optimal batch size is essential. However, it is important to note that the batch

size cannot be increased indefinitely, as the GPU may not be able to accommodate the tensors.

D. Power Consumption

Power consumption is one of the most important aspects when using deep learning frameworks. To measure power consumption in TensorFlow, we used the pynvml library, which supports power measurement on NVIDIA GPUs. Our objective was to compare power consumption for different batch sizes, but we found that batch size did not significantly affect power consumption as depicted in Figure 5. Data on power consumption is important for optimizing deep learning models for both performance and energy efficiency. With pynvml, we were able to accurately measure power consumption during our experiments, providing valuable insights for future optimization efforts.

E. Distributed Training: DataParallel vs. DistributedDataParallel

We found that DistributedDataParallel (DDP) performs better than DataParallel, although achieving performance speed-up is not trivial. DDP offers near-linear speed-ups with an increase in resource allocation. However, the programming complexity of DDP is higher compared to DataParallel, which requires minimal code changes for distributed training as can be seen in Listing 2 and Listing 1.

VII. Conclusion

Our study provides insightful information on how PyTorch and TensorFlow perform for training and inference tasks across a range of deep learning models, batch sizes, and input shapes. The results show no clear winner between the two frameworks; instead, the decision is largely influenced by the task, batch size, and model architecture. The effect of batch size on inference time was also observed, highlighting the significance of choosing the ideal batch size for effective model serving. Additionally, our research compared the trade-offs between DataParallel and DistributedDataParallel for distributed training in PyTorch, offering helpful guidance for practitioners in choosing the best strategy for their projects. Moreover, we also showed that DistributedDataParallel like Horovod provides near-linear speed-ups with an increase in resources.

VIII. Future Works

Based on our findings, future research can be focused in a number of areas to further improve our understanding of distributed training methods and deep learning framework performance:

- Analysis of Additional Frameworks: To provide a more thorough comparison and assist practitioners in making better choices when selecting a framework for their projects, we intend to expand our analysis to include additional deep learning frameworks, such as MXNet and CNTK.

```python
def train(rank, world_size):
  dist.init_process_group("nccl", rank=rank,
    world_size=world_size)
  device = torch.device("cuda:{}".format(rank))
  train_sampler = DistributedSampler(trainset,
    num_replicas=world_size, rank=rank)
  trainloader = DataLoader(trainset, batch_size=
    batch_size, sampler=train_sampler)
  model = ResNet50().to(device)
  model = DistributedDataParallel(model,
    device_ids=[rank])
  criterion = nn.CrossEntropyLoss()
  optimizer = optim.SGD(model.parameters(), lr
    =0.01, momentum=0.9)
  training_time = 0
  for epoch in range(epochs):
      start_time = time.time()
      model.train()
      train_loss = 0
      for inputs, targets in trainloader:
          inputs, targets = inputs.to(device),
    targets.to(device)
          optimizer.zero_grad()
          outputs = model(inputs)
          loss = criterion(outputs, targets)
          loss.backward()
          optimizer.step()

if __name__ == "__main__":
  world_size = torch.cuda.device_count()
  mp.spawn(train, args=(world_size,), nprocs=
    world_size, join=True)
```

Listing 1: Distributed training with DistributedDataParallel

```python
device = torch.device("cuda" if torch.cuda.
    is_available() else "cpu")
trainloader = DataLoader(trainset, batch_size=
    batch_size)
model = ResNet50().to(device)
model = DataParallel(model, device_ids=[0, 1])
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr
    =0.01, momentum=0.9)
training_time = 0
for epoch in range(epochs):
    start_time = time.time()
    model.train()
    train_loss = 0
    for inputs, targets in trainloader:
        inputs, targets = inputs.to(device),
    targets.to(device)
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, targets)
        loss.backward()
        optimizer.step())
```

Listing 2: Distributed training with DataParallel

- Investigating Impact of Hardware Configurations: We will gain a better understanding of how to optimize PyTorch and TensorFlow for various system setups, resulting in increased efficiency and resource utilization, by investigating the effects of various hardware configurations on the performance of these frameworks.
- Advanced Distributed Training Techniques: To improve performance and convergence at scale, we intend to delve into cutting-edge distributed training methods such as gradient compression and autotuning. We intend to further assist practitioners in optimizing their distributed training workflows by incorporating these techniques into our experiments.
- Integration of Domain-Specific Libraries and Accelerators: Examining the integration of domain-specific libraries, such as NVIDIA's cuDNN and TensorRT or AMD's ROCm, and accelerators, like Google's TPU, can reveal the performance gains and restrictions of each framework when making use of these specialized resources. This would make it possible to comprehend the frameworks' capabilities in environments with more specialized hardware.

By exploring these directions, our research could provide a more comprehensive and valuable resource for the deep learning community, helping practitioners make better-informed decisions when selecting frameworks, models, and training strategies.

References

[1] Paszke et al. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. Advances in Neural Information Processing Systems.

[2] Sergeev and Del Balso (2018). Horovod: Fast and Easy Distributed Deep Learning in TensorFlow. arXiv:1802.05799.

[3] NVIDIA. Caffe2 Deep Learning Framework 2017. https://developer.nvidia.com/caffe2.

[4] Banerjee, D.S., Hamidouche, K., & Panda, D.K., Re-Designing CNTK Deep Learning Framework on Modern GPU Enabled Clusters, 2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), 2016, pp. 144–151, DOI: 10.1109/CloudCom.2016.0036.

[5] Canziani, A., Paszke, A., & Culurciello, E. (2016). An Analysis of Deep Neural Network Models for Practical Applications. arXiv:1605.07678.

[6] Google, TensorFlow. [Online]. Available: https://www.tensorflow.org/.

[7] NVIDIA. GPU-Accelerated TensorFlow 2018. https://www.nvidia.com/en-us/data-center/gpu-acceleratedapplications/tensorflow/.

[8] Abadi M., et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems 2016. arXiv Prepr. arXiv1603.04467, 2016.

[9] Goldsborough. P., A tour of TensorFlow 2016. arXiv Prepr. arXiv1610.01178.

[10] Comparing Top Deep Learning Frameworks. https://deeplearning4j.org/compare-dl4j-tensorflow-pytorch.

[11] Torch. What is Torch? http://torch.ch/. [Online]. Available: http://torch.ch/.

[12] Chetlur S., et al. Cudnn: Efficient primitives for deep learning 2014. arXiv Prepr. arXiv1410.0759.

[13] Shi, S., Wang, Q., Xu, P., & Chu, X., Benchmarking state-of-the-art deep learning software tools 2016. In 7th International Conference on Cloud Computing and Big Data (CCBD), 2016, 99–104. DOI: 10.1109/CCBD.2016.029.

[14] Chollet. F. Deep learning with python, 2017 Manning Publications Co.

[15] Goodfellow I. J., et al. Pylearn2: a machine learning research library 2013. arXiv Prepr. arXiv1308.4214.

[16] Van Merriënboer B., et al. Blocks and fuel: Frameworks for deep learning 2015. arXiv Prepr. arXiv1506.00619.

[17] Shatnawi, A., Al-Bdour, G., Al-Qurran, R., & Al-Ayyoub, M. A comparative study of open source deep learning frameworks 2018. In 2018 9th International Conference on Information and Communication Systems (ICICS), 72- 77. DOI: 10.1109/I-ACS.2018.8355444.

[18] Measuring and Visualizing GPU Power Usage in Real Time with Asyncio and Matplotlib. https://danielmuellerkomorowska.com/2022/02/14/measuring-and-visualizing-gpu-power-usage-in-real-time-with-asyncio-and-matplotlib/