Touchstone: A Distributed Framework for TensorFlow Benchmarking

Rohit Keswani
Department of Computer Science
Golisano College of Computing and Information Sciences
Rochester Institute of Technology
Rochester, NY 14623
rohit.keswani@mail.rit.edu
Advisor: M. Mustafa Rafique <mrafique@cs.rit.edu>

***Abstract*—TensorFlow has been increasingly adopted by researchers, students, and professors for machine learning applications as it provides several functionalities for development and deployment in heterogeneous environments. Moreover, there is constant development around TensorFlow that makes it popular and accessible. However, currently, there is no benchmark framework for TensorFlow to provide performance statistics of workloads. In this paper, Touchstone is proposed as a Distributed Framework for TensorFlow Benchmarking to report statistics of models developed using TensorFlow. It also performs workload distribution across available resources for maximum resource utilization.**

***Index Terms*—Tensorflow; Benchmark; Neural Networks; Image Captioning; Workload Distribution**

I. Introduction

Machine Learning is a booming field. There have been many developments in recent times, and it is ever-evolving. Machine Learning pertains to training a computer to perform tasks and take decisions effortlessly, which otherwise have to be performed by humans. It is known that machine learning algorithms take a lot of time to perform tasks. One of the prominent reasons why these algorithms take a lot of computation time is the under-utilization of resources. A concept of distributed systems has been existent for a long time to distribute the computing requirements, perform computations on all available cores and machines. This idea can enable us to perform maximum utilization of the resources available to reduce the time required for machine learning applications. We can achieve maximum utilization of available resources by performing workload distribution [1].

Python has been proved to be quite promising for developing these applications. With the help of several libraries available with it, learning and developing machine learning models is comparatively less complex than before. Libraries such as PyTorch [2], Theano [3], TensorFlow [4] have eased out the development of machine learning models. Out of several libraries, TensorFlow [4] has emerged to be most popular amongst researchers because it provides excellent functionalities to perform parallel computations and is more flexible as it is a low-level library.

TensorFlow [4] is an open-source machine learning platform developed by Google Brains at Google. It is used to create, train, and test a variety of machine learning models. With the help of these models, machine learning students, professors, researchers are able to provide breakthrough discoveries with applications never imagined before. TensorFlow [4] provides with several functionalities that ease out the development of models in a parallel environment. It enables models to execute on CPU, GPU, or TPU as required by the developers. TensorFlow [4] is a machine learning system that operates on at large scale and in heterogeneous environments. However, there does not exist a standardized framework to gauge the performance of models as per hardware.

In this paper, Touchstone: A Distributed Framework for TensorFlow Benchmarking is proposed that would enable the researchers to fetch performance statistics of their models. It will prove to help understand the impact of change in hardware to the performance of the models. Moreover, it will perform workload distribution with the available resources that will result in better resource utilization and reduced run time.

The rest of the paper is organized as follows: Section II presents the background about TensorFlow [4], other libraries, and benchmarking suites in this domain. Section III presents the idea of the framework of Touchstone and it's architecture. Section IV details about the implementation strategies. Later, Section V presents the results and performance metrics of the framework. Further, Section VI and Section VII are about the lessons learned and the future scope of the Touchstone.

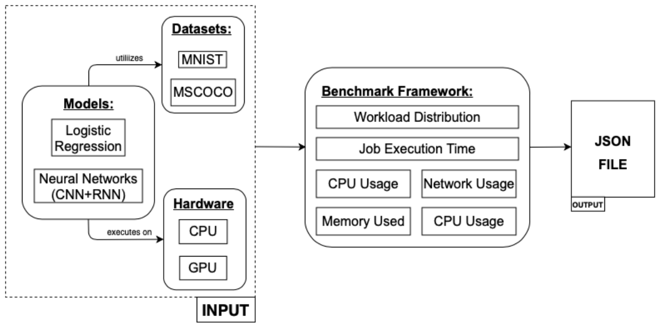

Fig. 1. Proposed Architecture for Touchstone

II. Background

TensorFlow [4] provides several additional functionalities compared to PyTorch [2]. One of the differences is with the origin of two libraries. TensorFlow [4] is developed by Google, whereas PyTorch [2] is developed by Facebook.

Both the libraries make use of directed acyclic graphs (DAG) for model creation. However, they differ in the type of graph. TensorFlow [4] creates a static graph that requires Python to create the graph of the model and then executes it. In PyTorch [2] dynamic graphs are created, which means you can modify the model as you go. Apart from this, Tensorflow Serving makes it a lot easier for deployments with Tensorflow, and TensorBoard [5] enables the researchers to display visualizations over a web server.

There has been a similar work that has been done with Apache Spark: SparkBench [6]. Since Apache Spark supports workloads such as Machine Learning, SQL Queries, Graph Computation, and Streaming Applications, SparkBench focuses on developing benchmark statistics for several applications based on these workloads. With the help of this, developers can compare their models, hardware performance with the one provided from SparkBench [6] for similar types and sizes of the dataset.

Since TensorFlow [4] supports only Machine Learning workloads, Touchstone will focus on developing a distributed benchmark suite for Machine Learning applications developed with TensorFlow [4]. With Touchstone, researchers would be able to gather performance statistics for the models and perform workload distribution according to the available resources.

III. Proposed Architecture

A. System Overview

Touchstone is a central entity in the architecture of the framework as shown in Figure 1. For the scope of the study, popular TensorFlow models such as Logistic Regression [7] and Neural Networks [8] are used. Popular and most-used datasets such as MNIST [9] and MSCOCO [10] are utilized by the aforementioned models. These models are executed on different types of hardware such as CPU and GPU. Execution of TensorFlow Models with datasets and on specific hardware is the input for Touchstone, which gathers all performance statistics and performs workload distribution to output a JSON file with all the details.

B. System Details

To dive deeper, in this section all the components of the architecture are explained in detail.

1) Datasets: There are two datasets in consideration for this paper. The datasets have been selected in order of their increasing size, usage, and popularity. Most of Machine learning applications begin with MNIST [9], and follow on to use large datasets such as MSCOCO [10] and ImageNet [11]. For this framework, we will be focusing on MNIST [9] and MSCOCO [10].

1) *MNIST* [9]: It is a dataset of handwritten images. It has a training set of 60,000 examples and a test set of 10,000 examples over 10 categories. The 10 categories are digits from 0 to 9. This dataset has a size of around 10MB. It is a good candidate as it does not require any pre-processing and formatting.
2) *MS-COCO* [10]: It is a large-scale object detection, segmentation, and captioning dataset.This dataset has a size of 16GB. It has around 330K images, with over 80 categories. It is a good candidate because a lot of models are implemented on MSCOCO [10] or with datasets around the size of this dataset.

2) Models: Following are the models implemented with Tensorflow and Tensorflow-GPU libraries. The models are selected based on usage and popularity.

- *Logistic Regression* [7]: It performs analysis on the data with the help of prediction and a cost function as shown in [12]. In this paper, we will be performing Logistic Regression [7] on MNIST [9] dataset mentioned in Section III-B1 to classify the digits from 0 to 9. The training parameters for the model are: learning_rate = 0.01, training_steps = 1000, batch_size = 256.
- *Neural Networks (VGG16)* [13]: Neural Network with VGG 16 architecture will be modeled on MSCOCO [10] dataset mentioned in Section III-B1. This model will be performing Image Captioning for images in MSCOCO [10] as described in [14] and [15]. The training parameters for the model are: learning_rate = 1e-3, batch_size = 512, epochs = 50.

3) Hardware: Aforementioned models in Section III-B2 would be executed on CPU and GPU.

1) *CPU*: The local machine in consideration for executing models on CPU has Darwin operating system with the processing capability of 2.3 GHz Dual-Core Intel Core i5 and memory of 8 GB 2133 MHz LPDDR3.
2) *GPU* [13]: GPU machine in consideration for executing models has Linux operating system with processing capability of Intel(R) Xeon(R) Silver 4214 CPU @ 2.20GHz with 24 cores. It has 10 Tesla T4 GPU's with 2560 NVIDIA CUDA Cores and a memory of 16 GB GDDR6.

4) Touchstone: Touchstone is the integral component for benchmarking TensorFlow [4]. It performs workload distribution and gathers performance statistics. It gathers the following statistics:

1) *Job Execution Time*: The time took by execution of model to train and test the data.
2) *Total CPU Usage*: It calculates the processing power consumed by model execution in terms of percentage of CPU used.
3) *Total CPU Usage/core*: It calculates the number of cores available in the hardware, and returns the usage of each core.
4) *Total Memory*: It calculates the memory used by the model for its execution.
5) *Total data sent over network*: Several models perform computations over the cloud, or gather dataset from remote repository. This calculates the data sent to gather the required details.
6) *Total data received over network*: For the data sent to collect the required details, there is data sent back. This calculates the data received to gather the required details.
7) *System Information*: It gathers the system details about the system that performs execution of the model. Following is the information gathered:
   a) *System*: Operating system of the hardware.
   b) *Node Name*: Name of the machine over network.
   c) *Release*: Operating system release number.
   d) *Version*: The version of the kernel.
   e) *Machine*: The architecture of the machine (32-bit or 64-bits).
   f) *Processor*: The processor of the hardware (i386 or i686, for 64 and 32 bit respectively).

It also performs workload distribution by distributing variables or parameters of the Model across all the available CPU/GPU or specific CPU/GPU assigned and aggregating them once the computation is completed. This enables in maximum resource utilization and reduction in the execution of the time. To be compatible with Touchstone to perform workload distribution [1], the program of the model has to be slightly modified. This modification requires developers to just add 2 lines of code, which can be referenced from the Readme file at RIT CS GIT

5) Output format: The output of the Touchstone is the JSON file as key-value pairs. For every gathered statistic, a key is created in a JSON file, and the value is the fetched statistic.

IV. Implementation

Touchstone is developed with Python and it supports workload distribution for models developed using TensorFlow [4]. It can gather performance statistics of models developed with any library as it uses default python library functionality for that. It is developed with around 200 lines of code and the code is available at RIT CS GIT.

Algorithm 1: Touchstone Pseudo-code

```
perform_workload_distribution = true;
for every model in models package do
    if perform_workload_distribution then
        //Modify model development code;
        1. Retrieve strategy scope before Model
          object instantiation.
    end
    1. Create a sub-process for model execution;
    2. Gather statistics of the sub-process;
    3. Store the statistics in a dictionary;
    4. Create a JSON file to hold a dictionary;
    Result: Save the JSON file.
end
```

The statistics are gathered with the help of "psutil" library of Python, which works with the help of "ps" System Command. Statistics are retrieved at the process level. For every model, a new subprocess is created which executes the model. Performance details of each model process present in the JSON file as shown in Section V is retrieved with "psutil" library. The System information is gathered with "platform" library, which is similar to using "uname" on CLI.

Workload Distribution [1] can be performed on any TensorFlow [4] model by adding as little as 2 lines of code. Workload distribution is performed with the help of TensorFlow's Mirrored Strategy [16]. This strategy uses one replica per device and sync replication for its multi-GPU version and it returns a number of replicas over which gradients are aggregated. With the help of this approach gradients are computed across all GPU cores and aggregated.

All the executions were performed from CLI to minimize the impact of background processes. The

TABLE I
Workloads executed with Touchstone

Model	Dataset	Hardware	Workload distribution
Logistic Regression	MNIST	CPU	false
Logistic Regression	MNIST	GPU	false
Neural Networks +VGG16	MSCOCO	GPU	false
Neural Networks +VGG16	MSCOCO	GPU	true

overview of Touchstone execution in presented in Algorithm 1. Additional models and datasets can be added and executed on various hardware to provide an array of benchmark statistics for researchers.

V. Results

Touchstone provides the output in the form of JSON files. It is evident from statistics of numerous executions the impact of hardware on the performance of the models. The impact of workload distribution was observed and it proved the goal that distributing computations across cores and devices, leads to lesser run time and better resources consumption. Four executions are performed with Touchstone and can be seen in Table I. Each row represents details about each execution.

Listing 1 details about the executions statistics of first execution mentioned in Table I. The key-value pairs of the JSON file has information as listed in Section III-B4.

Listing 1. Logistic Regression performed on MNIST dataset executed on a CPU.

```
{
  "Job Execution Time": "1.1438920577367147 minutes",
  "Total CPU Usage": "0.7%",
  "Total CPU usage/per core": "0.029166666666666664%",
  "Total Memory used by the model": "1.2421875 MB",
  "Total bytes sent over network": "95.41053295135498 MB",
  "Total bytes received over network": "96.00036144256592 MB",
  "System Information": {
    "System": "Darwin",
    "Node Name": "Rohits—MacBook—Pro—2.local",
    "Release": "19.4.0",
    "Version": "Darwin Kernel Version 19.4.0: Wed Mar 4 22:28:40
          PST 2020; root:xnu—6153.101.6~15/RELEASE_X86_64",
    "Machine": "x86_64",
    "Processor": "i386"
  }
}
```

Listing 2 details about the executions statistics of second execution mentioned in Table I. It can be seen by comparing Listing 1 and Listing 2 as shown in Figure 2, that with improvement in hardware details, the Job Execution Time reduced by 48%. Moreover, with increased number of cores, and processing capability as mentioned in Section III-B3 CPU usage and CPU usage/core tends to 0.0%.

Listing 2. Logistic Regression performed on MNIST dataset executed on a GPU.

```
{
  "Job Execution Time": "34.8992760181427 seconds",
  "Total CPU Usage": "0.0%",
  "Total CPU usage/per core": "0.0%",
  "Total Memory used by the model": "145.18359375 MB",
  "Total bytes sent over network": "26.0 KB",
  "Total bytes received over network": "58.0 KB",
  "System Information": {
    "System": "Linux",
    "Node Name": "lovegood",
    "Release": "4.15.0—88—generic",
    "Version": "#88—Ubuntu SMP Wed Mar 4 20:11:34 UTC 2020",
    "Machine": "x86_64",
    "Processor": "x86_64"
  }
}
```

Listing 3 mentions details about third execution from Table I. It performs Image Captioning on MSCOCO dataset. No workload distribution is performed here.

Listing 3. Image Captioning with Neural Networks+VGG16 performed on MSCOCO dataset executed on a GPU without workload distribution.

```
{
  "Job Execution Time": "11.2236111111 hours",
  "Total CPU Usage": "79.7%",
  "Total CPU usage/per core": "2.9583333333333335%",
  "Total Memory used by the model": "2.09952768 GB",
  "Total bytes sent over network": "2.8203331804 GB",
  "Total bytes received over network": "2.4315580590032912 GB",
  "System Information": {
    "System": "Linux",
    "Node Name": "lovegood",
    "Release": "4.15.0—88—generic",
    "Version": "#88—Ubuntu SMP Sun Mar 15 12:24:43 UTC 2020",
    "Machine": "x86_64",
    "Processor": "x86_64"
  }
}
```

Listing 4 mentions details about fourth execution from Table I. Here, workload distribution is performed. It can be seen from Listing 3 and Listing 4 as visualized in Figure 3 that with distribution of workload, job time reduced by almost 10%. Moreover, Figure III clearly shows it also led to increased resource utilization by approximately 2%.

Listing 4. Image Captioning with Neural Networks+VGG16 performed on MSCOCO dataset executed on a GPU with workload distribution.

```
{
  "Job Execution Time": "10.073084436323908 hours",
  "Total CPU Usage": "81.5%",
  "Total CPU usage/per core": "3.3958333333333335%",
  "Total Memory used by the model": "1.1968512 GB",
  "Total bytes sent over network": "1.1664712131023407 GB",
  "Total bytes received over network": "971.6640644073486 MB",
  "System Information": {
    "System": "Linux",
    "Node Name": "lovegood",
    "Release": "4.15.0—88—generic",
    "Version": "#88—Ubuntu SMP Mon Mar 16 18:35:23 UTC 2020",
    "Machine": "x86_64",
    "Processor": "x86_64"
```

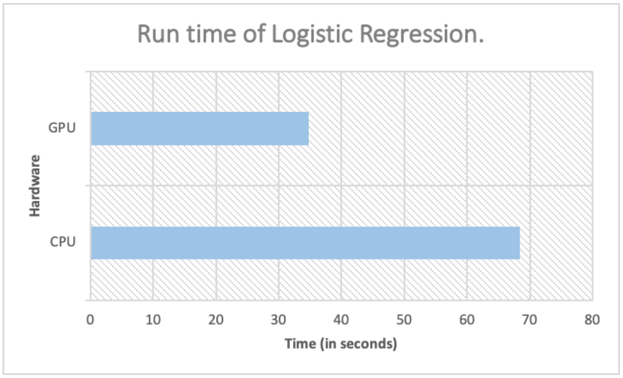

Fig. 2. Run Time Comparison for Logistic Regression

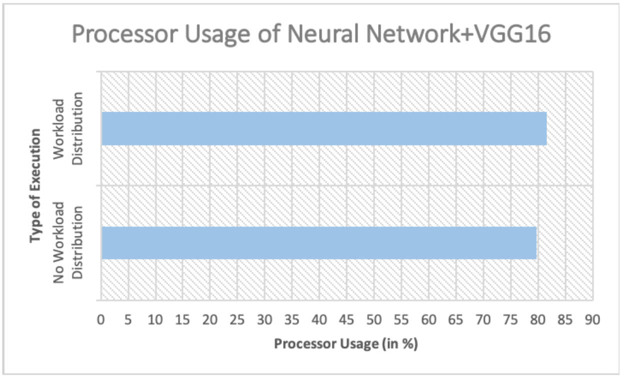

Fig. 4. Processor Usage Comparison for Neural Network + VGG16

```
  }
}
```

Figure 2 presents the graphical representation for run time of first and second execution from the Table I. The first execution from the table is Logistic Regression [7] performed on MNIST [9] dataset on the local machine. Second execution is Logistic Regression [7] performed on MNIST [9] dataset on the GPU. It is evident as the hardware improves, the execution run time of the model has reduced.

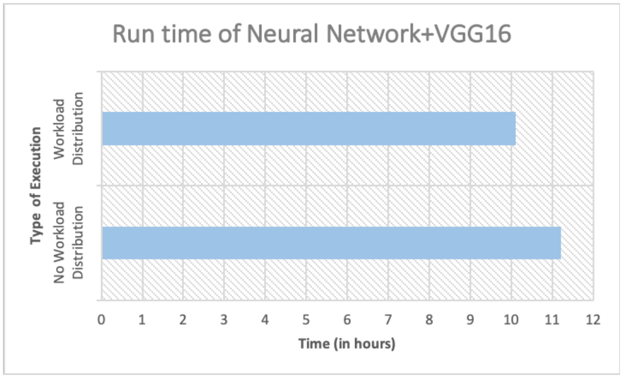

Fig. 3. Run Time Comparison for Neural Network + VGG16

Figure 3 presents the graphical representation for run time of third and fourth execution from the Table I. Third execution from the table is Neural Network + VGG16 model performed on MSCOCO [10] dataset to perform image captioning [15]. It is executed on GPU without workload distribution [1]. The fourth execution is Neural Network + VGG16 model performed on MSCOCO [10] dataset to perform image captioning [15] on GPU instance with workload distribution [1]. It is observed that with workload distribution, the job execution time of model reduced by almost 10%. Execution time dropped from ∼ 11 hours to ∼10 hours.

Figure 4 presents the graphical representation for processor usage of third and fourth execution from the Table I. Third and Fourth execution are detailed above. In Figure 4, it can be observed that with workload distribution, the processor usage of the hardware spiked by almost 2%. With this, it can be successfully stated that, workload distribution leads to better resources utilization and reduced job execution time.

VI. Conclusion

Touchstone accomplished the goals to gather run time statistics and perform workload distribution for the execution of models. Algorithm 1 explains well about the implementation details of Touchstone. In Section V, it can be evidently seen the reduction in run time by 48% between Listing 1 and Listing 2 with improvement in hardware capabilities. Also, in comparison of Listing 3 and Listing 4, it is seen run time reduced by almost 10% and resource utilization spiked by 2% on the same hardware with workload distribution.

With the help of Touchstone, it will be easier for researchers and organizations to understand the impact of hardware modification on the performance of models. Touchstone will be able to answer questions like:

1) How much time and memory does it take to perform Image Captioning on MSCOCO on Tesla GPU with 2560 cores, 16 GB GDDR6 RAM?
2) Should the organization go for 2496 cores, 12GB GPU, or 2048 cores 8GB TPU?

VII. Future Work

There is quite a scope for further development and enhancement of Touchstone. Several models, with different combinations of datasets, can be executed on various hardware to gather relevant statistics. Even more statistics can be gathered depending on the usability of researchers, and bottleneck identification can be done at each line of code. Efforts can be made in direction to perform workload distribution without any modifications.

References

[1] R. Alonso and L. L. Cova, “Sharing jobs among independently owned processors,” in *[1988] Proceedings. The 8th International Conference on Distributed*, 1988, pp. 282–288.

[2] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, “Pytorch: An imperative style, high-performance deep learning library,” in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8026–8037.

[3] T. Team, R. Al-Rfou, G. Alain, A. Almahairi, C. Angermueller, D. Bahdanau, N. Ballas, F. Bastien, J. Bayer, A. Belikov, A. Belopolsky, Y. Bengio, A. Bergeron, J. Bergstra, V. Bisson, J. Bleecher Snyder, N. Bouchard, N. Boulanger-Lewandowski, X. Bouthillier, and Y. Zhang, “Theano: A python framework for fast computation of mathematical expressions,” p. arXiv:1605.02688, May 2016.

[4] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, “Tensorflow: A system for large-scale machine learning,” in *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI’16. USA: USENIX Association, 2016, p. 265–283. [Online]. Available: https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf

[5] K. Wongsuphasawat, D. Smilkov, J. Wexler, J. Wilson, D. Mane, D. Fritz, D. Krishnan, F. Viegas, and M. Wattenberg, “Visualizing dataflow graphs of deep learning models in tensorflow,” *IEEE Transactions on Visualization and Computer Graphics*, vol. PP, pp. 1–1, 08 2017.

[6] M. Li, J. Tan, Y. Wang, L. Zhang, and V. Salapura, “Sparkbench: A spark benchmarking suite characterizing large-scale in-memory data analytics,” *Cluster Computing*, vol. 20, no. 3, p. 2575–2589, Sep. 2017. [Online]. Available: https://doi.org/10.1007/s10586-016-0723-1

[7] J. Peng, K. Lee, and G. Ingersoll, “An introduction to logistic regression analysis and reporting,” *Journal of Educational Research - J EDUC RES*, vol. 96, pp. 3–14, 09 2002.

[8] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, A. M. Umar, O. U. Linus, H. Arshad, A. A. Kazaure, U. Gana, and M. U. Kiru, “Comprehensive review of artificial neural network applications to pattern recognition,” *IEEE Access*, vol. 7, pp. 158 820–158 846, 2019.

[9] L. Deng, “The mnist database of handwritten digit images for machine learning research [best of the web],” *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[10] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollar, and L. Zitnick, “Microsoft coco: Common objects in context,” in *ECCV*. European Conference on Computer Vision, September 2014. [Online]. Available: https://www.microsoft.com/en-us/research/publication/microsoft-coco-common-objects-in-context/

[11] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

[12] “Logistic regression,” https://github.com/aymericdamien/TensorFlow-Examples, accessed: 2020-02-10.

[13] S. Liu and W. Deng, “Very deep convolutional neural network based image classification using small training sample size,” in *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*, 2015, pp. 730–734.

[14] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, “Show and tell: A neural image caption generator,” in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 3156–3164.

[15] “Image captioning with neural networks,” https://github.com/Hvass-Labs/TensorFlow-Tutorials, accessed: 2020-02-10.

[16] “Image captioning with neural networks,” https://www.tensorflow.org/api_docs/python/tf/distribute/MirroredStrategy, note = Accessed: 2020-02-10.