CSE 598 Data-Intensive Systems for Machine Learning – Spring 2024

# A Technical Report on Performance analysis of various datasets for Image Captioning on PyTorch and TensorFlow

Vatsal
1229611606
*Computer Science*
*Arizona State University*
Tempe, United States
vnolas48@asu.edu

Nishit Mukesh Patel
1229782140
*Computer Science*
*Arizona State University*
Tempe, United States
npate133@asu.edu

Keval Samir Shah
1225553979
*Computer Science*
*Arizona State University*
Tempe, United States
kshah80@asu.edu

***Abstract*—The task of image captioning has been widely explored in computer vision research; however, there is a notable lack of comprehensive comparative performance analysis across different libraries for image captioning models. This report aims to address this gap by conducting a comparative analysis of the performance between the TensorFlow and PyTorch frameworks on a large data set using a complex CNN-LSTM model. By solving this problem, we aim to provide valuable insights into the relative strengths and weaknesses of these frameworks for image captioning applications. Through rigorous experimentation and evaluation, we seek to facilitate informed decision-making regarding the choice of framework for image captioning tasks, thereby advancing the state-of-the-art in computer vision research.**

***Index Terms*—Machine learning, Image Captioning, PyTorch, Tensorflow.**

## I. Introduction

Machine learning for image captioning using PyTorch or TensorFlow represents a groundbreaking approach to understanding visual content. These frameworks leverage deep learning models, such as Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN), to automatically generate descriptive captions for images. PyTorch and TensorFlow offer powerful tools and libraries that streamline the development of image captioning models, making it accessible to both researchers and practitioners. With PyTorch's dynamic computation graph and TensorFlow's static graph execution, developers can choose the framework that best suits their preferences and requirements. By training on large data sets of images paired with corresponding captions, these models learn to associate visual features with textual descriptions, enabling applications in image indexing, content retrieval, and accessibility for visually impaired individuals. As machine learning continues to advance, the field of image captioning holds promise for enhancing human-computer interaction and understanding visual content in diverse domains.

In this study, we undertake a comparative analysis of PyTorch and TensorFlow frameworks for image captioning, focusing on Long Short-Term Memory (LSTM) networks. Our objective is to evaluate their performance across diverse datasets, notably Flickr8k, Flickr30k and MS COCO 2017, under consistent model architecture and hyper-parameters. To ensure impartiality, we maintain uniformity in architectural design and parameter configurations across both frameworks. Given the scarcity of pre-trained models tailored explicitly for image captioning, we embark on crafting our bespoke architecture, adopting identical hyper-parameters in both PyTorch and TensorFlow. By adopting this approach, we aim to provide an insightful comparison, shedding light on the relative strengths and weaknesses of each framework in the context of image captioning tasks. This study helps bridge the gap in existing literature, offering valuable insights into the practical implications of employing PyTorch and TensorFlow for image captioning applications.

## II. Problem Description

The problem of image captioning, though well-established, faces a critical gap in the form of comprehensive performance evaluations across various deep learning frameworks. Despite the rapid progress in this domain, the absence of systematic comparisons between leading frameworks such as PyTorch and TensorFlow remains conspicuous. This dearth of research significantly impedes informed decision-making regarding framework selection for image captioning tasks. The lack of empirical evidence on the relative strengths and weaknesses of PyTorch and TensorFlow in the context of image captioning poses a substantial challenge to practitioners and researchers alike. Consequently, there is a pressing need for rigorous comparative studies to elucidate the performance discrepancies between these frameworks and facilitate more informed decisions in the development of image captioning applications.

## III. Literature Review

Image captioning has garnered significant attention in recent years, with various approaches proposed to tackle the task effectively. Long Short-Term Memory (LSTM) networks have emerged as a popular choice for generating descriptive captions due to their ability to capture long-range dependencies in sequential data. LSTM networks, with their gated architecture, are particularly effective in modeling sequential data and have been successfully applied to image captioning tasks.

- **Theoretical Overview of LSTM:** Long Short-Term Memory (LSTM) networks are a type of recurrent neural network (RNN) designed to address the vanishing gradient problem in traditional RNNs. LSTMs utilize a memory cell and various gates (input, output, and forget gates) to selectively retain or forget information over time. This capability allows LSTMs to capture long-range dependencies in sequential data, making them suitable for tasks like image captioning.

- **How LSTM is used for Image Captioning:** In image captioning, LSTMs are used to generate descriptive captions for input images. The process involves encoding the image into a fixed-length feature vector using a convolutional neural network (CNN) and then decoding this feature vector into a sequence of words using an LSTM network. By training the model on paired image-caption datasets, the LSTM learns to associate visual features with corresponding textual descriptions, enabling it to generate accurate and contextually relevant captions for new images. [1].

The model that we'll be exploring is popularly known as the CNN-LSTM model outlined in the research paper titled "Learning CNN-LSTM Architectures for Image Caption Generation" and is specifically designed for sequence prediction problems for datasets with spatial features like images and videos. As explained above, This architecture involves using Convolutional Neural Network (CNN) layers for feature extraction on input data combined with LSTMs to perform sequence prediction on the feature vectors. In short, CNN LSTMs are a class of models that are both spatially and temporally deep and sit at the boundary of Computer Vision and Natural Language Processing. These models have enormous potential and are being increasingly used for many sophisticated tasks such as text classification, video conversion, and so on.

- **Effectiveness of LSTM:** LSTMs are effective for image captioning due to their ability to capture long-range dependencies and model sequential data effectively. [6] Unlike traditional RNNs, LSTMs can remember relevant information over longer sequences, resulting in more coherent and contextually relevant captions. Additionally, the gated architecture of LSTMs enables them to selectively retain or discard information, allowing for more precise control over the caption generation process.

Despite the effectiveness of LSTM networks, there is a notable lack of pre-trained models specifically tailored for image captioning. While pre-trained models exist for tasks like image classification and object detection, there is a gap in the availability of pre-trained models for image captioning. One notable exception is the BLIP (Bottom-Up and Top-Down Image Parser) model by Salesforce, but its TensorFlow implementation is not widely available or documented. Datasets such as Flickr8k, Flickr30k, and MS COCO have been widely used for training and evaluating image captioning models. [5]. These datasets provide a diverse range of images along with human-annotated captions, making them suitable for benchmarking different approaches to image captioning.

- **Brief introduction to Pytorch:** PyTorch is a widely-used open-source machine learning library renowned for its flexibility, simplicity, and dynamic computation graph capabilities. It offers a seamless experience for both research and production-level deployment of deep learning models. One of its standout features is its dynamic computation graph, allowing for intuitive and agile model development through imperative programming. PyTorch's elegant and Pythonic interface enables rapid prototyping and experimentation, making it a favorite among researchers and practitioners alike. Its strong community support, extensive documentation, and integration with popular libraries such as NumPy further contribute to its appeal. With its emphasis on simplicity, PyTorch has become a go-to framework for building state-of-the-art neural networks across various domains, from computer vision and natural language processing to reinforcement learning and beyond.

- **Brief introduction to Tensorflow:** TensorFlow 2.0 represents a significant evolution of the popular deep learning framework, focusing on simplicity, ease of use, and flexibility. One of the key features introduced in TensorFlow 2.0 is eager execution by default, which allows for immediate execution of operations, making it easier to debug and understand code. Additionally, TensorFlow 2.0 provides a more intuitive and high-level API through tf.keras, enabling developers to quickly build and train deep learning models with fewer lines of code. Moreover, TensorFlow 2.0 integrates seamlessly with other popular libraries and frameworks, facilitating interoperability and enabling developers to leverage a wide range of tools and resources. With improvements in performance, usability, and compatibility, TensorFlow 2.0 has solidified its position as a leading platform for deep learning research and application development.

In our literature review, we did not come across previous research specifically comparing the performance of image captioning between PyTorch and TensorFlow. However, we did find several studies that conducted comparisons on image classification tasks using various deep learning frameworks. One such study presented experimental results demonstrating the performance of different frameworks on a common image classification benchmark dataset.

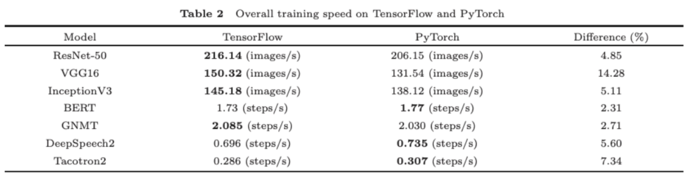


**Table 2** Overall training speed on TensorFlow and PyTorch

| Model | TensorFlow | PyTorch | Difference (%) |
|---|---|---|---|
| ResNet-50 | **216.14** (images/s) | 206.15 (images/s) | 4.85 |
| VGG16 | **150.32** (images/s) | 131.54 (images/s) | 14.28 |
| InceptionV3 | **145.18** (images/s) | 138.12 (images/s) | 5.11 |
| BERT | 1.73 (steps/s) | **1.77** (steps/s) | 2.31 |
| GNMT | **2.085** (steps/s) | 2.030 (steps/s) | 2.71 |
| DeepSpeech2 | 0.696 (steps/s) | **0.735** (steps/s) | 5.60 |
| Tacotron2 | 0.286 (steps/s) | **0.307** (steps/s) | 7.34 |

Fig. 1. Comparative performance analysis between PyTorch and TensorFlow on training popular models.

## IV. Experimental Setup

The experimental setup is designed to test and compare the performance of PyTorch and TensorFlow using a CNN-LSTM architecture for image captioning. The aim is to train a consistent model architecture for both the frameworks with consistent hyper-parameters, same hardware and datasets.

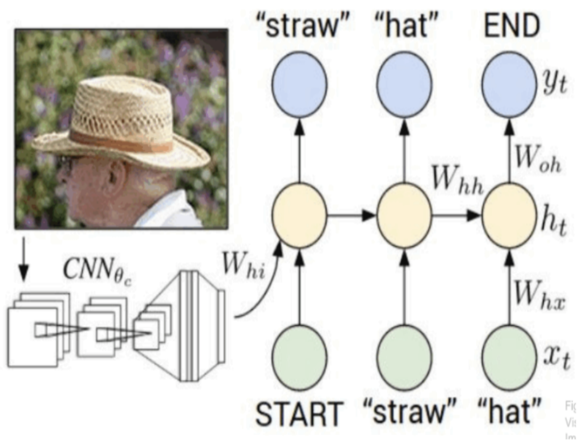


Fig. 2. High level architecture design

The program execution as per the given model architecture warrants two major steps:

- **Feature Extraction:** We use the popular pre-trained model DenseNet-201, which is trained on the popular data set **ImageNet** for extracting the features from all the images in the dataset. DenseNet-201 is a highly efficient convolutional neural network (CNN) architecture of 201 layers known for its dense connectivity patterns. These connections facilitate better feature reuse and gradient flow during training, resulting in more effective feature representation learning. By leveraging the hierarchical representations learned by DenseNet-201 on large-scale image datasets, the model can capture intricate details and semantic information necessary for generating accurate captions. The dense connectivity within DenseNet-201 enables efficient information propagation, allowing the model to comprehend both local and global dependencies within images. Overall, integrating DenseNet-201 enhances the model's capability to describe complex visual scenes with precision.
- **Decoding using LSTM:** After that, we put an LSTM decoder into practice and train it to produce captions using the features that were extracted in the earlier stage. Using the context gathered from the image features and the words that have already been generated, the LSTM decoder learns to anticipate the next word in the caption. The decoder LSTM efficiently captures the linguistic structure and semantics of the captions through this sequential generation process, guaranteeing that the output text closely matches the visual content of the input image. The decoder may also be modified to employ attention mechanisms, which would let it to concentrate on various areas of the picture while producing each word. This would improve the model's capacity to provide precise and contextually appropriate captions. All things considered, the decoder LSTM functions as the linguistic analogue of the CNN's visual feature extraction, bridging the semantic gap between visual information and textual description in the image captioning task.

In order to get a conclusive and comprehensive result, we run the above model training on a variety of hardware

- **Google Colab CPU:** This is the least computationally powerful hardware chosen by us and the purpose for choosing it is to get the raw CPU performance analysis for both the frameworks.
  - CPU: Intel Xeon CPU @ 2.20GHZ
  - Memory: 12.7 GB
  - Storage: 108 GB SSD
- **Google Colab GPU:** This is the standard tesla T4 GPU model provided by Google Colab. The purpose of choosing this hardware is to get the raw baseline GPU performance analysis for both the frameworks.
  - GPU: Nvidia Tesla T4
  - GPU Memory: 15 GB
  - Storage: 108 GB SSD
- **ASU SOL CPU:** We also trained our models using the SOL supercomputer at ASU using the most powerful CPU available. This compute environment provides performance analysis of both frameworks on the most powerful CPU available
  - CPU: AMD EPYC 7413 24-core processor
  - Memory: 16 GB
  - Storage: 1 TB SSD
- **ASU SOL GPU:** We also trained our models using the SOL supercomputer at ASU using the most powerful GPU available. This compute environment pro-

vides performance analysis of both frameworks in high-performance computing environments
  - GPU: Nvidia A100
  - GPU Memory: 80 GB
  - Storage: 1 TB SSD

In order to achieve our objectives we have incorporated usage of several softwares and tools to achieve the best and most accurate results:

- **Python [v3.10.12]:** Python is the preferred choice of programming language due to ease of prototyping and support for both TensorFlow and PyTorch.
- **TensorFlow [v2.15.0]:** TensorFlow 2.0 is used in the default eager mode to execute the training.
- **PyTorch [v2.2.1 + cu121]:** PyTorch latest version is used in the default eager mode to execute the training.
- **TorchVision [v0.17.1 + cu121]:** TorchVision is a computer vision library in PyTorch, providing tools and utilities for tasks like image preprocessing, dataset handling, and pre-trained models for tasks such as image classification, object detection, and segmentation.
- **Pandas [v2.0.3]:** Pandas is a Python library used for data manipulation and analysis, offering powerful data structures and functions for handling structured data, such as tabular data and time series. It's widely used for tasks like data cleaning, transformation, and exploration, making it an essential tool for data scientists and analysts.
- **Scalene [v1.5.19]:** Scalene is a high-performance CPU, GPU and memory profiler for Python that does a number of things that other Python profilers do not and cannot do. It runs orders of magnitude faster than many other profilers while delivering far more detailed information. It is also the first profiler ever to incorporate AI-powered proposed optimizations.

## V. Dataset Description

For the purpose of analysis, we have chosen a set of 3 popular image datasets of varying size and varieties for training purpose

- **Flickr8k:** The Flickr8k is the smallest dataset out of all and has the following specifications:
  - 8,092 images
  - Five captions per image
  - 256 X 256 pixels per image
- **Flickr30k:** The Flickr30k has the following specifications:
  - 30,000 images
  - Five captions per image
  - 256 X 256 pixels per image
- **MS COCO 2017:** The MS COCO dataset by Microsoft is the largest dataset we tested and has the following specifications:
  - 118,287 images
  - 591,753 captions
  - Varying image sizes with some exceeding 400 X 400 in size

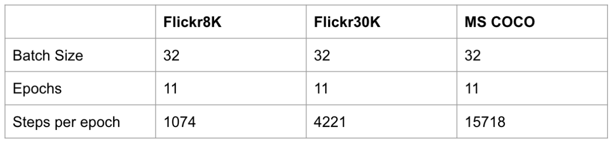
## VI. Methodology

As described in great detail in the experimental setup, our CNN-LSTM model architecture consists of two major steps i.e. feature extraction and training the LSTM for generating captions. As a part of the first step, we explicitly extract the most important features from all the images in the dataset and then use the extracted features to train the LSTM model for caption generation.
We measure performance benchmarks for both the libraries based on the below parameters.

- **Throughput:** Throughput helps us understand the amount of work done by the machine in a given unit of time. It is a direct measure of the speed of computation and provides us with an early estimate of the amount of time required to complete the job. We define throughput as number of images processed in a given unit of time.
- **Memory Consumption:** Memory consumption is an important parameter for identifying how efficient a framework is in terms of resource utilization. A higher memory consumption indicates a potentially bad performance and may lead to a bottleneck. A lower memory consumption indicates that the given framework is more efficient with managing the available resources and effectively frees up resources not needed anymore.
- **Total Training Time:** Total training is the most conclusive parameter as it gives us the direct measure of the time taken to complete a job. The main goal of this paper is to provide with this estimate across a variety of hardware and datasets to provide an accurate result.

We tested the Flickr8k data set on both the SOL supercomputer and Google Colab using PyTorch and TensorFlow. For the rest of the datasets i.e. Flickr30k and MS COCO, we tested them only on the SOL supercomputer using both frameworks.

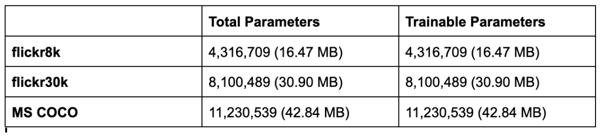
The decided hyperparameters for all the datasets can be found in the referenced image below



| | Flickr8K | Flickr30K | MS COCO |
|---|---|---|---|
| Batch Size | 32 | 32 | 32 |
| Epochs | 11 | 11 | 11 |
| Steps per epoch | 1074 | 4221 | 15718 |

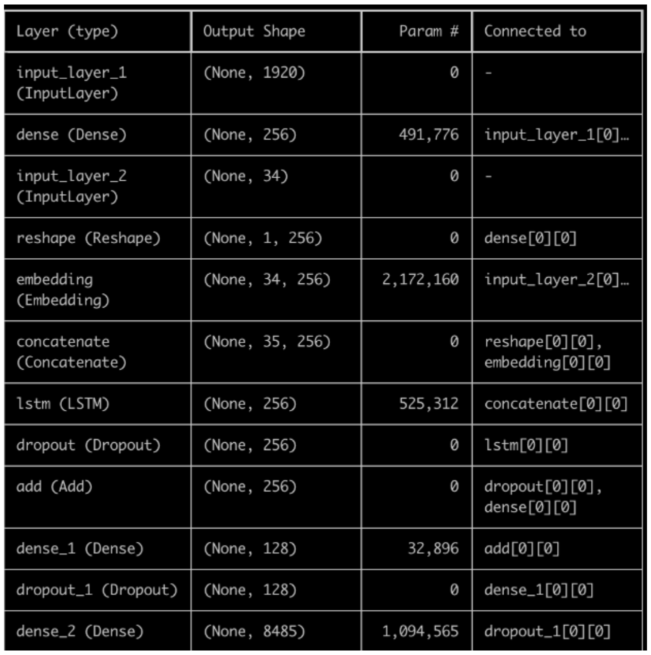
Fig. 3. Hyperparameters chosen for each dataset

In the following, we can find the structures of the decoder model for all the data sets and get a detailed analysis of the number of training parameters per data set and the model sizes per data set. We have a consistent model architecture and size for both the PyTorch and TensorFlow frameworks.



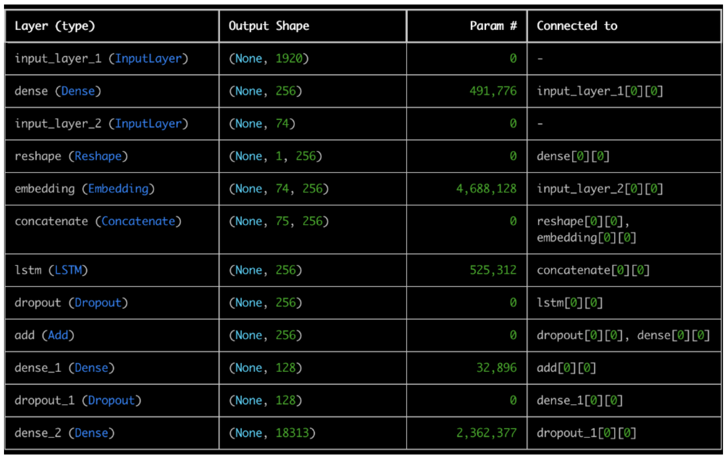
| | Total Parameters | Trainable Parameters |
|---|---|---|
| **flickr8k** | 4,316,709 (16.47 MB) | 4,316,709 (16.47 MB) |
| **flickr30k** | 8,100,489 (30.90 MB) | 8,100,489 (30.90 MB) |
| **MS COCO** | 11,230,539 (42.84 MB) | 11,230,539 (42.84 MB) |

Fig. 4. LSTM-model Parameter Counts for the three datasets chosen



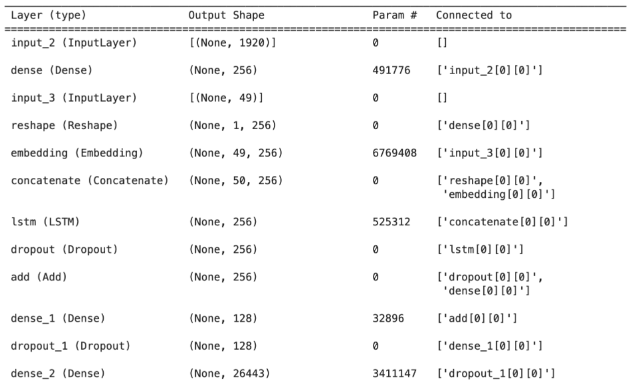
| Layer (type) | Output Shape | Param # | Connected to |
|---|---|---|---|
| input_layer_1 (InputLayer) | (None, 1920) | 0 | - |
| dense (Dense) | (None, 256) | 491,776 | input_layer_1[0]… |
| input_layer_2 (InputLayer) | (None, 34) | 0 | - |
| reshape (Reshape) | (None, 1, 256) | 0 | dense[0][0] |
| embedding (Embedding) | (None, 34, 256) | 2,172,160 | input_layer_2[0]… |
| concatenate (Concatenate) | (None, 35, 256) | 0 | reshape[0][0], embedding[0][0] |
| lstm (LSTM) | (None, 256) | 525,312 | concatenate[0][0] |
| dropout (Dropout) | (None, 256) | 0 | lstm[0][0] |
| add (Add) | (None, 256) | 0 | dropout[0][0], dense[0][0] |
| dense_1 (Dense) | (None, 128) | 32,896 | add[0][0] |
| dropout_1 (Dropout) | (None, 128) | 0 | dense_1[0][0] |
| dense_2 (Dense) | (None, 8485) | 1,094,565 | dropout_1[0][0] |

Fig. 5. Model structure for Flickr8k dataset



| Layer (type) | Output Shape | Param # | Connected to |
|---|---|---|---|
| input_layer_1 (InputLayer) | (None, 1920) | 0 | - |
| dense (Dense) | (None, 256) | 491,776 | input_layer_1[0][0] |
| input_layer_2 (InputLayer) | (None, 74) | 0 | - |
| reshape (Reshape) | (None, 1, 256) | 0 | dense[0][0] |
| embedding (Embedding) | (None, 74, 256) | 4,688,128 | input_layer_2[0][0] |
| concatenate (Concatenate) | (None, 75, 256) | 0 | reshape[0][0], embedding[0][0] |
| lstm (LSTM) | (None, 256) | 525,312 | concatenate[0][0] |
| dropout (Dropout) | (None, 256) | 0 | lstm[0][0] |
| add (Add) | (None, 256) | 0 | dropout[0][0], dense[0][0] |
| dense_1 (Dense) | (None, 128) | 32,896 | add[0][0] |
| dropout_1 (Dropout) | (None, 128) | 0 | dense_1[0][0] |
| dense_2 (Dense) | (None, 18313) | 2,362,377 | dropout_1[0][0] |

Fig. 6. Model structure for Flickr30k dataset



```
Layer (type)                  Output Shape          Param #   Connected to
===========================================================================================
input_2 (InputLayer)          [(None, 1920)]        0         []

dense (Dense)                 (None, 256)           491776    ['input_2[0][0]']

input_3 (InputLayer)          [(None, 49)]          0         []

reshape (Reshape)             (None, 1, 256)        0         ['dense[0][0]']

embedding (Embedding)         (None, 49, 256)       6769408   ['input_3[0][0]']

concatenate (Concatenate)     (None, 50, 256)       0         ['reshape[0][0]',
                                                               'embedding[0][0]']

lstm (LSTM)                   (None, 256)           525312    ['concatenate[0][0]']

dropout (Dropout)             (None, 256)           0         ['lstm[0][0]']

add (Add)                     (None, 256)           0         ['dropout[0][0]',
                                                               'dense[0][0]']

dense_1 (Dense)               (None, 128)           32896     ['add[0][0]']

dropout_1 (Dropout)           (None, 128)           0         ['dense_1[0][0]']

dense_2 (Dense)               (None, 26443)         3411147   ['dropout_1[0][0]']
```

Fig. 7. Model structure for MS COCO dataset

## VII. Results

After completing the training using both TensorFlow and Pytorch conclusively on various datasets and hardwares, we received definitive results for performance. For feature extraction since we use pre-trained models, we do not get a comparative analysis for training in both frameworks. However, we were able to attain benchmarking results across different sets of hardware for different datasets.

| | Flickr8K on GPU (Sol) | Flickr8K on GPU (Colab) | Flickr30K on GPU | MSCOCO on GPU |
|---|---|---|---|---|
| Avg. It/s | 13 | 9 | 12 | 10 |
| Total Time (mins) | ~10 | ~14 | ~41 | ~240 |

Fig. 8. Performance across different hardware for feature extraction of images using DenseNet-201

As we can see, MS COCO takes the highest time for feature extraction as expected given a huge dataset size.

We obtained a conclusive comparative analysis between PyTorch and TensorFlow on training the LSTM decoder for caption generation. While training, we achieved a validation loss of 3.6% on each execution for all our setups.

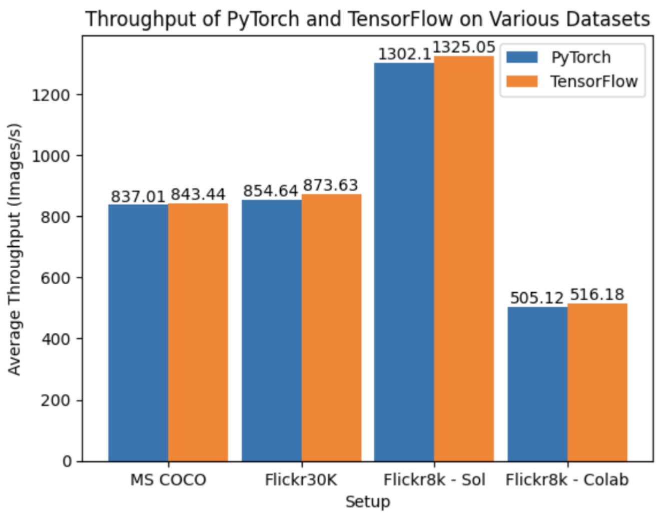


Fig. 9. Comparative analysis of images/sec for PyTorch and TensorFlow

As we can see, TensorFlow has a marginally higher throughput/second than PyTorch across all data sets and hardware. The highest difference can be observed for Flickr8K data set on SOL supercomputer where TensorFlow throughputs an average of 23 images more than PyTorch per second. Flickr8k dataset has consistently sized images of 256 X 256 dimensions and thus provides a reliable measure to indicate our result.

We can notice in Figure 9. that PyTorch has slightly lesser GPU memory consumption than TensorFlow. This result is expected as TensorFlow has a higher throughput indicating that the memory consumption would be higher. However, it could also mean that PyTorch has a better memory management

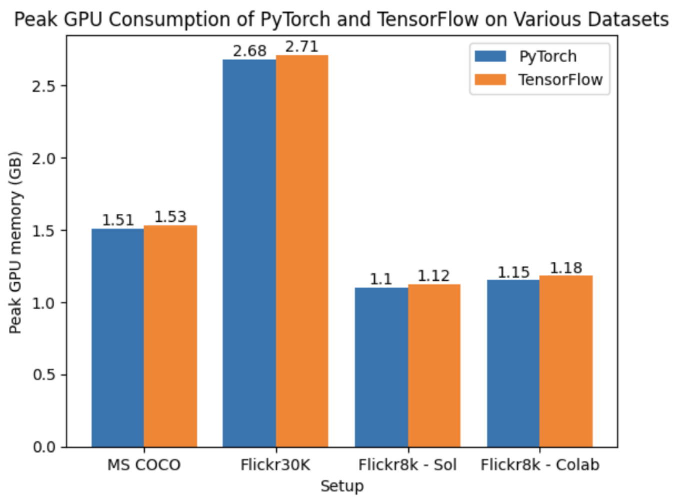


Fig. 10. Comparative analysis of GPU memory consumption for PyTorch and TensorFlow

overall and that it may lead to a better performance for even larger dataset due to better resource utilization!

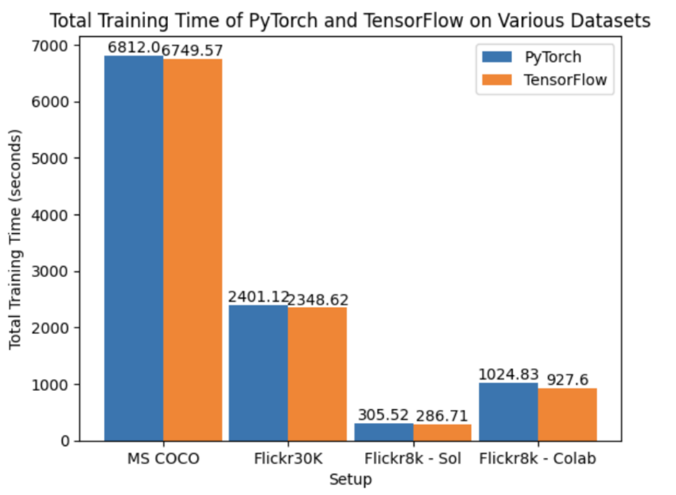


Fig. 11. Comparative analysis of Total Training Time for PyTorch and TensorFlow

As we can see in Figure 10. TensorFlow is marginally faster than PyTorch in terms of training completion time across all hardware and data sets. This result is inline with our expectations as TensorFlow had a higher throughput as well.

## VIII. Conclusion

In this project, we have conducted experimental validation of performance for TensorFlow and PyTorch in Image Captioning application using CNN-LSTM model. When comparing the frameworks, we can conclude that both have a very closely similar performance with TensorFlow having slightly better metrics. We attribute this result to the fact that these tools offload most of the computation to the same version of the cuDNN and cuBLAS libraries. The decision to choose either frameworks cannot be made on the basis of performance as there is no significant difference observed between the two. Therefore, a decision to use either frameworks can be made based on the qualitative user-experience and user comfort in using either frameworks. For this study, we found the qualitative user experience to be better with TensorFlow due to its integration with easy-to-use Keras API. Although we found TensorFlow to perform better qualitatively in our tests, we encountered an issue where it would randomly freeze during the middle of an epoch when using a GPU. This issue occurred only once across all our tests, and other users have reported experiencing the same problem. [15]

## IX. Future Work

Future work for this project could involve testing the same data sets on a distributed data architecture and incorporation of data-parallelism. We also plan on trying this same experiment with different batch sizes to check if we are able to find any further evidence supporting our results. A better approach would be to extend this experiment to use a larger data set such as the Google Captions dataset containing over 30 million images. We also plan to extend the experiments to the task of inferencing for image captioning.
An interesting avenue lies in the fact that PyTorch has a slightly better GPU memory management than TensorFlow. Thus, it is possible that the results might be different for larger datasets where PyTorch may potentially perform better due to efficient memory management than TensorFlow. Given the opportunity, we would be inclined to perform the same experiment for the Google Captions dataset using better computational results.

## References

[1] O. Vinyals, A. Toshev, S. Bengio and D. Erhan, "Show and tell: A neural image caption generator," 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, USA, 2015, pp. 3156-3164, doi: 10.1109/CVPR.2015.7298935. https://arxiv.org/abs/1411.4555.

[2] Learning CNN-LSTM Architectures for Image Caption Generation. Moses Soh, Department of Computer Science, Stanford University. https://cs224d.stanford.edu/reports/msoh.pdf.

[3] Sharma, P., Ding, X., Goodman, S., & Soricut, R. (2018). Conceptual captions: A cleaned,hypernymed, image alt-text dataset for automatic image captioning. https://aclanthology.org/P18-1238.pdf.

[4] An Empirical Study of Language CNN for Image Captioning. Jiuxiang Gu, Gang Wang, Jianfei Cai, Tsuhan Chen; Proceedings of the IEEE International Conference on Computer Vision(ICCV), 2017. https://arxiv.org/pdf/1612.07086.pdf.

[5] Lin, T. Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., ... & Zitnick, C. L. (2014). Microsoft coco: Common objects in context. In European conference on computer vision (pp. 740-755). Springer, Cham

[6] Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhutdinov, R., ... & Bengio, Y. (2015). Show, attend and tell: Neural image caption generation with visual attention. In Proceedings of the 32nd International Conference on Machine Learning (Vol. 37, pp. 2048-2057)

[7] Karpathy, A., & Fei-Fei, L. (2015). Deep visual-semantic alignments for generating image descriptions. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 3128-3137).

[8] Aganja, A., Upadhyaya, C., Gansi, O., & Suwal, S. (2021). Image captioning using CNN and Deep Stacked LSTM.

[9] Paszke et al., “PyTorch: An Imperative Style, High-Performance Deep Learning Library,” arXiv (Cornell University), vol. 32, pp. 8026–8037, Jan. 2019, [Online]. Available: https://arxiv.org/pdf/1912.01703.pdf

[10] M. Abadi et al., “TensorFlow: A system for large-scale machine learning,” arXiv (Cornell University), Jan. 2016, doi: 10.48550/arxiv.1605.08695.

[11] A. Agrawal et al., “TensorFlow Eager: A Multi-Stage, Python-Embedded DSL for Machine Learning,” arXiv (Cornell University), Feb. 2019, [Online]. Available: https://arxiv.org/pdf/1903.01855

[12] Dai, H., Peng, X., Shi, X. et al. Reveal training performance mystery between TensorFlow and PyTorch in the single GPU environment. Sci. China Inf. Sci. 65, 112103 (2022). https://doi.org/10.1007/s11432-020-3182-1

[13] A. Jain, A. A. Awan, H. Subramoni and D. K. Panda, ”Scaling TensorFlow, PyTorch, and MXNet using MVAPICH2 for High-Performance Deep Learning on Frontera,” 2019 IEEE/ACM Third Workshop on Deep Learning on Supercomputers (DLS), Denver, CO, USA, 2019, pp. 76-83, doi: 10.1109/DLS49591.2019.00015. keywords: Training;Throughput;Deep learning;Parallel processing;Program processors;Neural networks;Synchronization

[14] A. Jain, A. A. Awan, Q. Anthony, H. Subramoni and D. K. D. Panda, ”Performance Characterization of DNN Training using TensorFlow and PyTorch on Modern Clusters,” 2019 IEEE International Conference on Cluster Computing (CLUSTER), Albuquerque, NM, USA, 2019, pp. 1-11, doi: 10.1109/CLUSTER.2019.8891042. keywords: Training;Parallel processing;Computational modeling;Graphics processing units;Multicore processing;Central Processing Unit;DNN Training;Performance Characterization;MVAPICH2 MPI;TensorFlow;PyTorch;Horovod

[15] “GPU freezes randomly while training using tf.keras models,” Stack Overflow. https://stackoverflow.com/questions/59126952/gpu-freezes-randomly-while-training-using-tf-keras-models