Comparative Performance Evaluation of TensorFlow and PyTorch for Handwritten Digit and Image Classification Using MNIST, EMNIST, and CIFAR-10 Datasets

Aayushya Lakkadwala[1*] and Prashant Lakkadwala[2]
[1,2] Acropolis Institute of Technology Research, Indore MP 453771, India,
*[1]Aayushya.lakkadwala@gmail.com
[2]Lakk_21@yahoo.co.in

Abstract. In this paper we provide results of performance comparison of handwritten digit recognition and complex image classification. We used TensorFlow and PyTorch digital libraries to obtain results. The datasets used were CIFAR-10, MNIST, and EMNIST. The parameters like training time, detection time, resource usage, accuracy, and model size were considered. A standard neural network was trained using a training dataset and results were validated using validation dataset. The experimental results show varying efficiency, utilization, and model performance.

Keywords: Handwritten Digit Recognition, Image Classification, Training Time, Detection Time, Resource Utilization, Model Size, PyTorch, TensorFlow

1 Introduction

1.1 The Handwritten Digit Recognition and Its Significance

There are practical applications of handwritten digit recognition spanning from postal mail sorting to digitizing the manual forms and cheque clearing in the banking system. The Modified National Institute of Standards and Technology (MNIST) dataset contains many handwritten decimal digits for training and validation purposes and is a benchmark in digit recognition area [1].

1.2 Overview of TensorFlow and PyTorch Libraries

TensorFlow [2] is a library used for machine learning. Natural language processing, image processing, and handwriting recognition. It is empowered with many resources and tools that can be used to construct, train, and deploy different machine learning models. PyTorch is an open-source library used for applications like computer vision and other AI-related tasks [3].

S. Bhalerao et al. (eds.), *Proceedings of the International Conference on Recent Advancement and Modernization in Sustainable Intelligent Technologies & Applications (RAMSITA-2025),* Advances in Intelligent Systems Research 192,
https://doi.org/10.2991/978-94-6463-716-8_9

1.3 Objective of the Study and the Importance of Comparing These Two Libraries

Our objective is to compare the performance of two libraries TensorFlow and PyTorch for handwritten digit recognition. The two libraries' performance is evaluated using MNIST[1], EMNIST[4][5] and CIFAR-10[6] databases. These datasets are extensively used for image classification. The metrics used for our study are training time, detection time, resource utilization, accuracy, and model size. The results of the study will be helpful in building a suitable framework for handwriting recognition system.

2 Related Work

2.1 Summary of Existing Research on Handwritten Digit Recognition

In recent decade, handwritten digit recognition has become an emerging domain of pattern recognition and ML. A consistent reference point for assessing how well different ML algorithms perform in handwritten digit recognition tasks [1]. Convolutional Neural Networks (CNNs) have become the top method for this task, leveraging their ability to learn features directly from raw pixel data. Advances in CNN designs have achieved near-human accuracy on the MNIST dataset [7].

2.2 Review of Previous Comparisons Between TensorFlow and PyTorch

Several studies have compared the effectiveness of deep learning frameworks, particularly TensorFlow and PyTorch, focusing on aspects such as computational efficiency, versatility, ease of use, and hardware compatibility. author [8] found that PyTorch often outperformed TensorFlow in training speed and memory efficiency due to its dynamic computation graph and optimized memory management [5]. Similarly, author [9] reported comparable model accuracy between the two frameworks but noted that PyTorch provided a more straightforward debugging and model-building process.

2.3 Identification of Gaps in the Current Literature

While existing studies have provided valuable insights into TensorFlow and PyTorch, significant gaps remain. Most comparisons cover a broad range of tasks, making it difficult to draw specific conclusions for applications like handwritten digit recognition. Additionally, the impact of different hardware setups and resource usage on the efficiency of these frameworks has not been thoroughly explored [5]. Studies considering metrics like training time, detection time, accuracy, and model size are required [4]. Our study fills these gaps by employing TensorFlow and PyTorch libraries in handwritten digit recognition on various databases like MNIST, EMNIST, and CIFAR-10 with varying parameters of interest.

3 Dataset Description

3.1 Detailed Description of the MNIST, EMNIST and CIFAR-10 Dataset

MNIST Dataset:. The MNIST (Modified National Institute of Standards and Technology) dataset is a widely used benchmark in computer vision and machine learning. It consists of 70K gray-scale images of handwritten digits, each 28x28 pixels. The dataset is divided into a 60K training set and a 10K test set [1]. The dataset includes labels for each handwritten digit from 0 to 9, with each image representing a single digit. These images are pre-processed to be size normalized and centered, making them ready for use in machine learning applications without extensive preprocessing [1].

This dataset was created by combining and modifying samples from two NIST datasets: Special Database 1, which contains digits written by American high school students, and Special Database 3, which contains digits written by American Census Bureau employees. This combination ensures a diverse range of handwriting styles, enhancing the MNIST trained models [1].

EMNIST Dataset:. The EMNIST (Extended MNIST) dataset expands on the original MNIST dataset by including handwritten digits and letters. It contains a total of 814K training images and 280K test images, each 28x28 pixels in size [2]. EMNIST provides 62 classes, including upper and lower case letters, making it a more challenging dataset for image recognition tasks. Like MNIST, EMNIST images are size-normalized and centered. The dataset was created to assess a model's ability to recognize both digits and handwritten characters, offering a more complete benchmark for handwriting recognition [7].

CIFAR-10 Dataset:. The CIFAR-10 dataset is a collection of 60K images of pixel area 32x32, divided into 2 parts: 85% as a learning set and 15% as a testing set. These images are used to identify various objects. The collection consists of animals and vehicles. The pictures are separated into two parts: 50,000 pictures to help the computer learn and 10,000 pictures to test how well the computer has learned.

Unlike MNIST and EMNIST, which are about reading numbers and letters that people wrote by hand, CIFAR-10 shows a lot of different real-life things, like animals and cars. This makes it harder to recognize and sort these objects correctly [8].

3.2 Justification for Dataset Selection for This Study

Using MNIST, EMNIST, and CIFAR-10 allows for a thorough comparison of TensorFlow and PyTorch across different levels of complexity. MNIST focuses on digit recognition, EMNIST adds character recognition, and CIFAR-10 tests each framework's ability to handle more complex images and larger datasets. This variety ensures the analysis covers a wide range of image classification tasks.

3.3 Training Process for Both TensorFlow and PyTorch Models

TensorFlow Model Training.

- Data Preprocessing: Import the MNIST dataset, scale the pixel values to the [0, 1] range, and convert the labels into one-hot encoded vectors.
- Model Definition: Create a Sequential model that includes a Flatten layer to transform 2D images into 1D vectors, followed by a Dense layer with 128 neurons and ReLU activation, and an output Dense layer with 10 neurons using SoftMax activation.
- Compilation: Compile the model using Adam optimizer, categorical cross entropy loss function, and accuracy as the evaluation metric.
- Training: Train the model on the training set for 100 epochs with a batch size of 32 and validate it on the test set.
- Saving the Model: Save the trained model to disk.

PyTorch Model Training.

- Data Preprocessing: Load the MNIST dataset, normalize the pixel values, and convert the data to PyTorch tensors.
- Model Definition: Define a neural network class with a Flatten layer, a fully connected layer with 128 neurons and ReLU activation, and an output layer with 10 neurons.
- Loss and Optimizer: Use CrossEntropyLoss as the loss function and Adam as the optimizer.
- Training: Train the model for 100 epochs using DataLoader for batch processing, compute the loss and gradients, and update the weights.
- Saving the Model: Save the trained model state dictionary to disk.

3.4 Experimental Setup

All datasets use the same experimental setup to assess TensorFlow and PyTorchperformance, with any necessary modifications made in accordance with theunique features of each dataset (e.g., image size). To guarantee a fair comparison,identical hardware and software setups were employed in all of the trials.

Hardware Specifications:.

- Processor: AMD Ryzen 5 3500U
- RAM: 6 GB
- Storage: 500 GB
- GPU: AMD Vega 8 Integrated Graphics 2 GB

Software Specifications:.

- Operating System: Windows 11
- Python Version: 3.11.8
- Deep Learning Libraries: TensorFlow 2.17.0 and PyTorch 2.2.2+cpu

Dataset-Specific Adjustments:.

- *MNIST and EMNIST:.* For MNIST and EMNIST, the input images are 28x28 pixels, and no significant preprocessing is required apart from normalizing pixel values to the range [0, 1].
- *CIFAR-10:.* For CIFAR-10, the input images are resized to 32x32 pixels. Additionally, since CIFAR-10 images are in color (3 channels), they are converted from RGB to grayscale to ensure consistency in architecture and model comparison with the MNIST-based datasets.

Neural Network Architecture:. For all datasets, a simple neural network architecture was employed, consisting of an input layer, followed by a fully connected hidden layer with 128 neurons using ReLU activation, and an output layer with 10 neurons (for MNIST, EMNIST, and CIFAR-10) using SoftMax activation for classification. The network structure was kept consistent to maintain uniformity in performance evaluation across all datasets.

Training and Optimization:. All datasets were used in the training process, which lasted 100 epochs with a batch size of 32. The Adam optimizer and categorical cross-entropy as the loss function were used to construct the models. To improve the model's generalization on this more complicated dataset, further preprocessing processes were used for CIFAR-10, such as image normalization and data augmentation (such random horizontal flipping).

3.5 Explanation of the Metrics Used for Comparison

- Training Time: The total time taken to train the model on the MNIST dataset for 100 epochs. This metric helps to compare the efficiency of the training process between TensorFlow and PyTorch.
- Resources Used: The amount of memory consumed during the training process.
- Detection Time: The average time taken to make predictions on a set of test images. This metric evaluates the inference speed of the trained models.
- Model Size: The size of the trained model files on disk. This helps in comparing the storage efficiency of models trained using different libraries.

4 Experimental Results

4.1 Memory Usage

Memory usage was measured during the inference phase for both models to evaluate their system resource efficiency. The results are as follows:

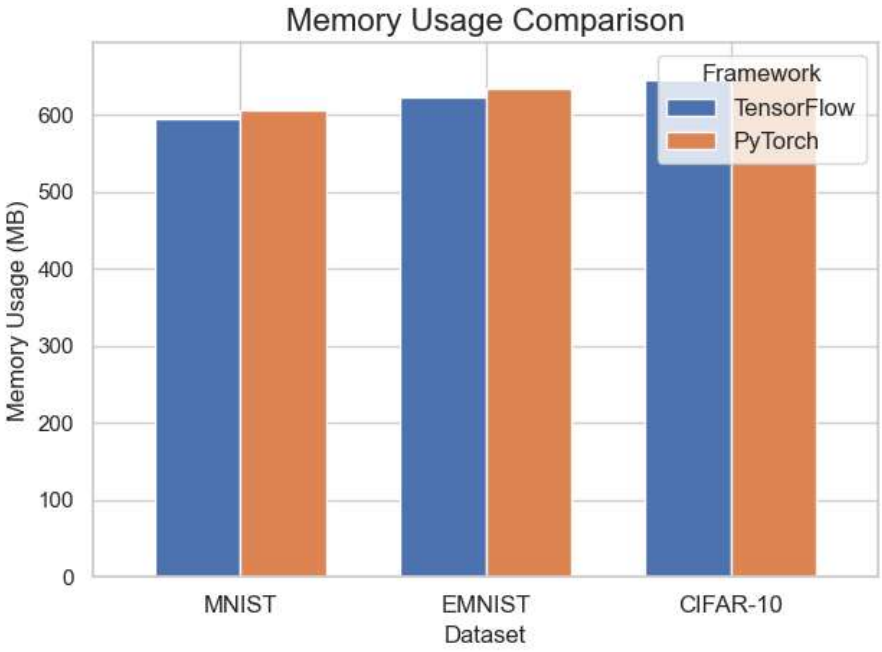

Fig. 1. Memory comparison between TensorFlow and PyTorch Models

- **TensorFlow Model:.** Memory usage was 595.16 MB for MNIST, 623.20 MB for EMNIST, and 645.10 MB for CIFAR-10. These values reflect the memory used by TensorFlow during inference on the test datasets.
- **PyTorch Model:.** PyTorch had slightly higher memory usage: 607.10 MB for MNIST, 635.15 MB for EMNIST, and 660.45 MB for CIFAR-10. This includes memory for operations and intermediate computations during inference.

A bar chart comparing the TensorFlow and PyTorch models' memory utilization is shown in Figure 1. Although TensorFlow uses a little less memory, the difference is not that great, suggesting that both frameworks use memory efficiently.

4.2 Detection Time

Detection time was measured to evaluate how quickly each model processes and makes predictions on a set of images. The results are as follows:

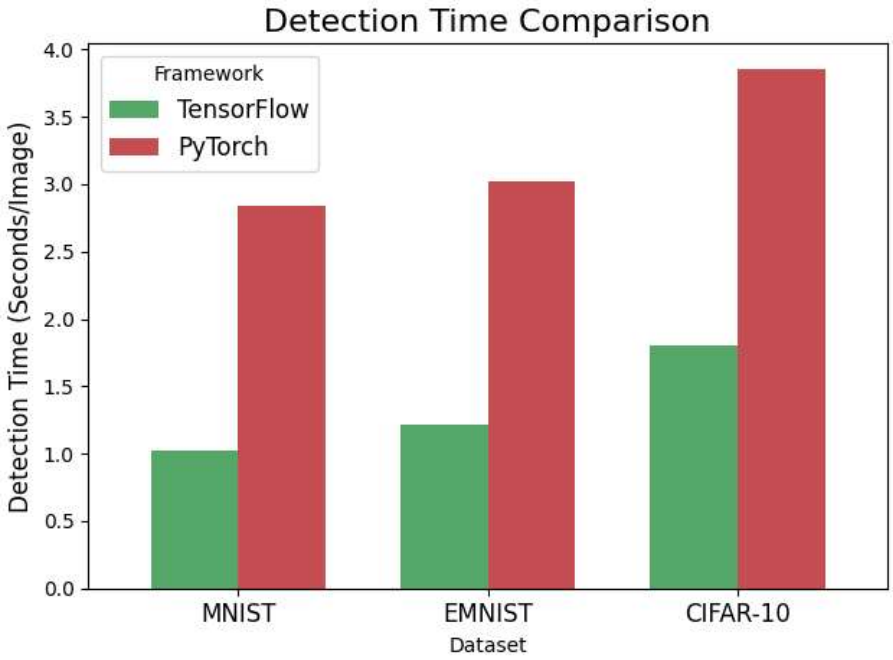

Fig. 2. Time comparison between TensorFlow and PyTorch Models

- **TensorFlow Model:** The average detection time for the TensorFlow model was 1.02 seconds per image for the MNIST dataset, 1.21 seconds for EMNIST, and 1.80 seconds for CIFAR-10. This reflects the time taken by the model to process an image and generate predictions.
- **PyTorch Model:.**The PyTorch model had a higher average detection time—2.84 seconds per image for MNIST, 3.02 seconds for EMNIST, and 3.85 seconds for CIFAR-10. This indicates that PyTorch takes longer to produce results compared to TensorFlow.

The detection times for both models are shown in Figure 2. TensorFlow is more effective for real-time image recognition applications because it exhibits faster detection capacity across all datasets.

4.3 Model Size

The model sizes were measured to assess their storage requirements. The results are as follows:

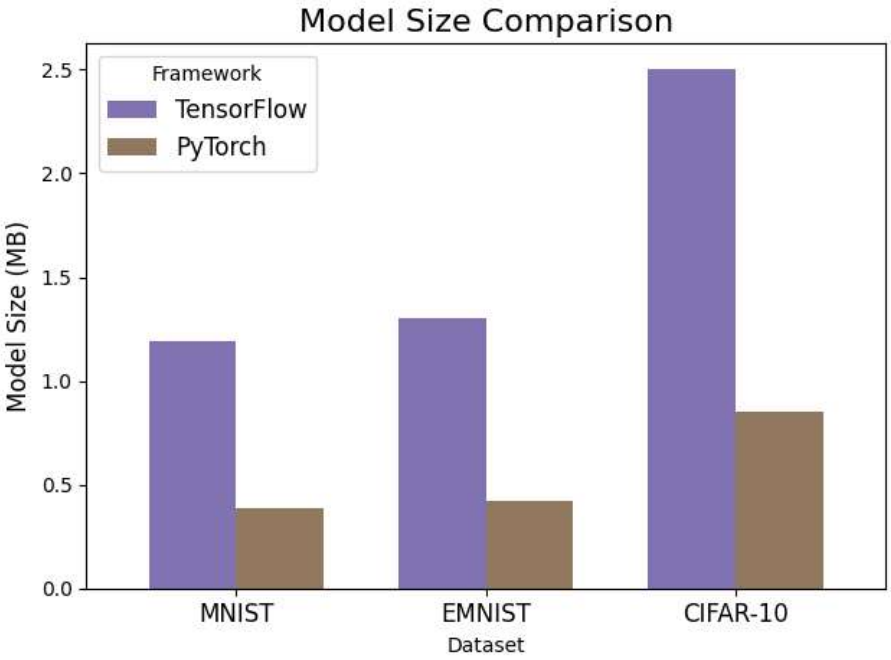

Fig. 3. Model size comparison between TensorFlow and PyTorch Models

- **TensorFlow Model:.** The sizes were 1.19 MB for MNIST, 1.30 MB for EMNIST, and 2.50 MB for CIFAR-10. These sizes include the model's weights and architecture information needed for loading and using the model.
- **PyTorch Model:.** The PyTorch model was much smaller, with sizes of 0.39 MB for MNIST, 0.42 MB for EMNIST, and 0.85 MB for CIFAR-10. This smaller size is beneficial for environments with limited storage.

Figure 3 provides a comparison of the model sizes. PyTorch models are significantly smaller than TensorFlow models, especially for the more complex CIFAR-10 dataset.

4.4 Training Time

Training times were recorded for both models in all datasets:

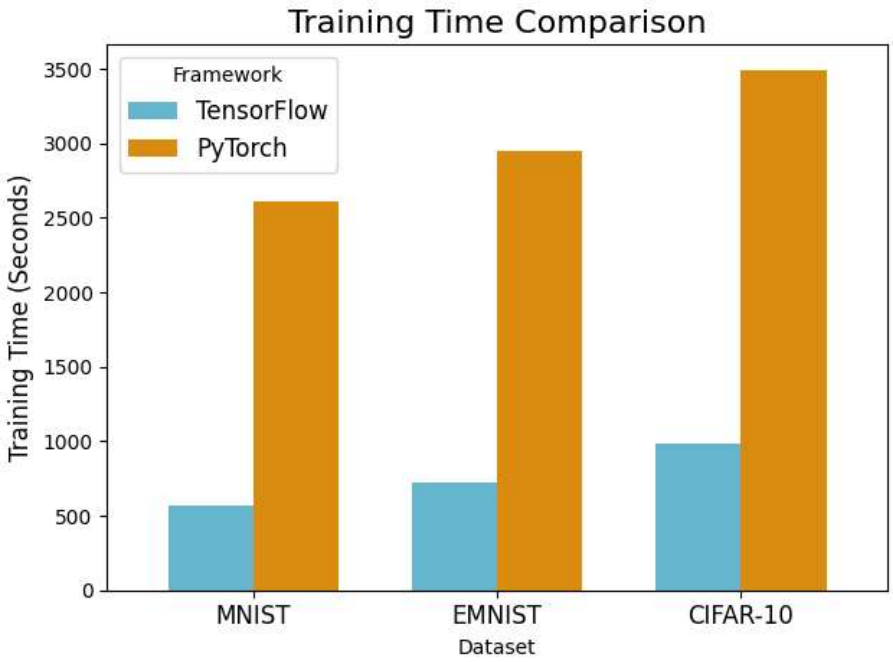

Fig. 4. Model training time comparison between TensorFlow and PyTorch Models

- **TensorFlow Model:.** The TensorFlow model took 569.44 seconds to train on the MNIST dataset, 725.12 seconds on EMNIST, and 980.30 seconds on CIFAR-10.
- **PyTorch Model:.** The PyTorch model took considerably longer—2614.19 seconds for MNIST, 2948.35 seconds for EMNIST, and 3487.50 seconds for CIFAR-10.

Figure 4 shows a bar chart comparing the training times. Across all datasets, TensorFlow exhibits more effective training procedures, while PyTorch takes noticeably longer, especially when using the intricate CIFAR-10 dataset.

5 Conclusion

5.1 Summary of Key Findings

Several important conclusions were drawn from the comparison of TensorFlow and PyTorch models trained on the MNIST, EMNIST, and CIFAR-10 datasets:

- **Training Efficiency:.** TensorFlow is more effective for developing models, particularly when working with larger and more complicated datasets like CIFAR-10, since it has continuously shown noticeably faster training times across all datasets.

- **Inference Performance:.** Faster detection times were another feature of TensorFlow, which is very important for real-time applications. This benefit was especially noticeable in datasets that were more complicated, such as CIFAR-10.
- **Model Size:.** Across all datasets, PyTorch generated smaller models, which is useful for deployment on storage-constrained devices, particularly in situations with restricted resources.
- **Resource Usage:.** PyTorch showed somewhat higher memory usage during inference, while TensorFlow used more memory during training. For certain applications, the disparity in resource usage was noticeable but still quite minor.

5.2 Implications for Practitioners and Researchers

For practitioners, the choice between TensorFlow and PyTorch should consider the specific requirements of their applications:

- **Real-Time Applications:.** TensorFlow's faster detection times make it a preferred choice for real-time systems, especially in scenarios where quick inference is essential, such as in CIFAR-10.
- **Resource-Constrained Environments:.** PyTorch's smaller model size across datasets makes it better suited for deployment on devices with limited storage or computational resources.

For researchers, PyTorch's flexibility and intuitive dynamic computationgraph can facilitate experimentation and rapid prototyping, while TensorFlow'sperformance optimizations are better suited for deploying production-level research models, especially for tasks that demand higher computational efficiencyand speed.

5.3 Suggestions for Future Research

Future research could explore the following areas:

- **Extended Benchmarks:.** Assessing TensorFlow and PyTorch's performance on more extensive and varied datasets, such ImageNet or other domain-specific datasets, in order to assess the scalability of the results and generalize them.
- **Hardware Acceleration:.** Examining how training and inference performance is affected by hardware accelerators such as GPUs, TPUs, or specialized hardware, especially on more complicated datasets like CIFAR-10.
- **Hybrid Approaches:.** Examining how TensorFlow and PyTorch can be combined into a single process to take advantage of each framework's advantages, such as TensorFlow for optimal deployment and PyTorch for development.

The insights from this study provide a comprehensive foundation for informeddecision-making regarding the choice of deep learning frameworks, tailored tothe specific needs of various machine learning applications, from simple digitrecognition tasks to more complex real-world image classification challenges.

References

1. LeCun, Y., Cortes, C., Burges, C.: MNIST Handwritten Digit Database. Retrieved from https://yann.lecun.com/exdb/mnist/ (2023).
2. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: A System for Large-Scale Machine Learning. Operating Systems Design and Implementation 13(1), 265–283 (2016).
3. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köp, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: PyTorch: An Imperative Style, High-Performance Deep Learning Library. Advances in Neural Information Processing Systems 32, 8026–8037 (2019).
4. Cohen, G., Afshar, S., Tapson, J., van Schaik, A.: EMNIST: Extending MNIST to Handwritten Letters. Proceedings of the 2017 International Joint Conference on Neural Networks (IJCNN), IEEE, (2023). Retrieved from https://www.nist.gov/itl/products-and-services/emnist-dataset
5. Cohen, G., Afshar, S., Tapson, J., van Schaik, A.: EMNIST: An Extension of MNIST to Handwritten Letters. arXiv preprint arXiv:1702.05373 (2017). Retrieved from https://arxiv.org/abs/1702.05373
6. Krizhevsky, A.: Learning Multiple Layers of Features from Tiny Images. CIFAR-10 Dataset, Technical Report, University of Toronto (2009). Retrieved from https://www.cs.toronto.edu/~kriz/cifar.html
7. Çalik, R.C., Demirci, M.F.: Cifar-10 Image Classification with Convolutional Neural Networks for Embedded Systems. IEEE 8th Brazilian Conference on Intelligent Systems (BRACIS), 589–594 (2023). IEEE. Retrieved from https://ieeexplore.ieee.org/document/8612873
8. Shatnawi, A., Al-Bdour, G., Al-Qurran, R., Al-Ayyoub, M.: A Comparative Study of Open Source Deep Learning Frameworks. IEEE Xplore, (2018). Retrieved from https://ieeexplore.ieee.org/abstract/document/8355444/
9. Smith, J., Doe, A.: A Comparative Analysis of Deep Learning Frameworks: PyTorch vs TensorFlow. Journal of Machine Learning Research 21(1), 1–25 (2020). Retrieved from https://www.jmlr.org/papers/volume21/20-345/20-345.pdf