Comparative Analysis of Neural Network Training Using NumPy and PyTorch on the MNIST Dataset

Dhimas Prayogi, Muhammad Iqbal

Abstract

This study discusses a comparative analysis between artificial neural network training using NumPy and PyTorch in classifying digits in the MNIST dataset. These two approaches are compared to evaluate effectiveness in terms of accuracy, computational efficiency, and ease of implementation. The NumPy model is manually built with two layers, utilizing basic numerical operations for the forward and backward propagation processes, while the PyTorch model uses a Multilayer Perceptron (MLP) architecture with torch.nn and torch.optim libraries. The training results show that PyTorch excels in stability and training time, as well as delivering consistently higher accuracy than NumPy. Visualization through Principal Component Analysis (PCA) shows a more regular distribution of digit clusters, while decision boundary mapping between digits 3 and 8 shows a clearer class separation in the PyTorch model. This study shows that while manual implementation with NumPy is beneficial for learning basic neural network concepts, the use of PyTorch is more recommended for real-world applications that require efficiency and scalability. These results are expected to be a reference in the selection of machine learning tools according to the needs and resources of developers.

Keywords: *Neural Network, NumPy, PyTorch, MNIST, Digit Classification*

Dhimas Prayogi[1]
[1]Information Technology, Universitas Pembangunan Panca Budi, Indonesia
e-mail: dhijay02@gmail.com[1]

Muhammad Iqbal[2]
[2]Information Technology, Universitas Pembangunan Panca Budi, Indonesia
e-mail: muhammadiqbal@dosenunpab.ac.id[2]
2nd International Conference on Islamic Community Studies (ICICS)
Theme: History of Malay Civilisation and Islamic Human Capacity and Halal Hub in the Globalization Era
https://proceeding.pancabudi.ac.id/index.php/ICIE/index

Introduction

The development of machine learning and deep learning technologies has played an important role in improving the accuracy and efficiency of various modern computing applications, particularly in the field of image classification (Judijanto et al., 2024). One of the most widely used datasets for testing classification algorithms is MNIST, which contains images of handwritten digits with a resolution of 28x28 pixels (Nocentini et al., 2022). This dataset became the de facto standard in the learning and testing of artificial neural network models (Rana et al., 2023). Although frameworks such as TensorFlow and PyTorch have provided efficient means for model training, understanding the internal mechanisms of neural network training is still an important need, especially among students and novice researchers (Joseph et al., 2021) ; (Novac et al., 2022). Therefore, model training approaches using basic numerical libraries such as NumPy are still relevant to deepen understanding of basic algorithms such as forward propagation, backward propagation, and gradient descent-based optimization (Zeng & Long, 2022).

The main problem raised in this study is how to thoroughly compare simple neural network training using NumPy and modern frameworks such as PyTorch, both in terms of accuracy, training time efficiency, implementation complexity, and computational resource needs (Antunes & Hill, 2024). Often, the machine learning approach only focuses on the final achievement in the form of accuracy without evaluating the learning process, the dissemination of data in the vector space, or the clarity of decision boundaries between classes (Jouini et al., 2024). To address this problem, the study offers a comprehensive Python project solution that integrates the implementation of a two-layer neural network using NumPy from scratch and a multilayer perceptron (MLP) model using PyTorch, complete with a variety of visualizations and automated reports (Beohar & Rasool, 2021).

The project was designed to train neural network models using both approaches on the MNIST dataset, record the value of loss and accuracy at each epoch, and compare the results (Yapıcı & Topaloğlu, 2021). In addition, data distribution visualization was carried out with the Principal Component Analysis (PCA) technique to project MNIST data into two dimensions, making it easier to understand the distribution of classes. To describe the classification boundaries between digits, a logistic regression model was used that was trained on the PCA reduction data to produce a visualization of decision boundaries, especially for digits 3 and 8. All of these metrics, graphs, and analysis results are then summarized in one automated report document in PDF format. This approach is expected to provide added value in the form of a complete practical, exploratory, and educational computing experience.

This research is based on several previous studies, (Gridin, 2022) compare the performance of deep learning frameworks such as TensorFlow and PyTorch in image classification, highlighting training efficiency and architectural flexibility. Deep (Gope et al., 2021) analyze the effectiveness of deep learning models in the classification of MNIST digits with various frameworks. Research (Prasetyo et al., 2024) building neural networks from scratch using NumPy as an educational medium to explicitly explain the process of forward and backward propagation. Deep (NA, 2023) examine the visualization of classification results using PCA techniques to understand the distribution of classes in two-dimensional spaces. Meanwhile, (Holl & Thuerey, 2024) perform a performance evaluation of a simple neural network based on NumPy and compare it with PyTorch in the context of educational and training efficiency. Based on the results of the review, no studies have been found that explicitly compare these two approaches directly in the context of integrated visual and educational projects, especially those that include PCA visualization, decision boundary, and documentation of results in the form of PDF automated reports (Okatan, 2023). This shows that there is a research gap that wants to be filled in this study.

The purpose of this study was to conduct a comparative analysis of simple neural network training using NumPy and PyTorch on the MNIST dataset. The main focus of the research includes the evaluation of model accuracy, convergence speed, complexity of code

implementation, efficiency of memory usage, and ease of interpretation of results through visualization. It is hoped that the results of this study can be a practical and educational reference for academics and novice practitioners in choosing the neural network training approach that best suits their needs, while emphasizing the importance of integrating numerical and visual evaluation in machine learning systems.

Literature Review

The following literature review examines the methodologies, performance metrics, and educational value of training neural networks using basic numerical libraries versus modern deep learning frameworks, specifically focusing on the MNIST dataset.

2.1. Framework Comparisons and Performance

Recent research has heavily focused on the efficiency and scalability of different deep learning frameworks. **Antunes and Hill (2024)** explored the reproducibility and energy efficiency of machine learning implementations across **Python, NumPy, TensorFlow, and PyTorch**, highlighting the variations in performance. Similarly, **Gridin (2022)** and **Yapıcı & Topaloğlu (2021)** emphasized that frameworks like **PyTorch** and **TensorFlow** offer superior architectural flexibility and training efficiency compared to manual implementations. Studies specifically comparing **PyTorch** and **TensorFlow** in the context of Convolutional Neural Networks (CNNs) have further validated their application efficiency in complex image classification tasks.

2.2. Educational Value of Manual Implementation

Despite the dominance of high-level frameworks, building neural networks from scratch using **NumPy** remains a vital educational tool. **Joseph et al. (2021)** and Novac et al. (2022) argue that understanding internal mechanisms—such as **forward propagation**, **backward propagation**, and **gradient descent**—is essential for students and novice researchers. Prasetyo et al. (2024) successfully used NumPy-based neural networks to explicitly explain these core processes. This manual approach, while less efficient for large-scale applications, provides deep insight into the mathematical foundations of machine learning.

2.3. The MNIST Dataset as a Benchmark

The **MNIST dataset**, featuring 28x28 pixel handwritten digits, continues to serve as the "de facto standard" for testing and learning artificial neural networks. Researchers such as **Gope et al. (2021)** and Rana et al. (2023) have utilized this dataset to analyze the effectiveness of various models and compare error rates across different machine learning architectures. Its standardized nature allows for clear benchmarking of accuracy and convergence speed between different training methodologies.

2.4. Data Visualization and Interpretation

Beyond final accuracy scores, the visual interpretation of data distribution and model decision-making is a growing area of interest. **Principal Component Analysis (PCA)** is frequently employed to reduce 784-dimensional MNIST data into 2D space for cluster analysis. **NA (2023)** demonstrated the use of PCA alongside SVM algorithms to understand class distribution. Furthermore, visualizing **decision boundaries**—particularly between visually similar digits like **3 and 8**—helps identify critical areas where models are prone to misclassification.

2.5. Research Gap and Synthesis

While existing literature covers framework performance and educational NumPy implementations separately, there is a lack of integrated studies that directly compare the two through a combined lens of numerical metrics and advanced visualizations like **decision boundary mapping** and **automated reporting**. This study addresses that gap by evaluating

NumPy and **PyTorch** not only on accuracy but also on implementation complexity and visual interpretability, providing a comprehensive reference for both academics and practitioners

Research Methodology

This study uses a computational experimental approach to compare the performance of simple neural network training using two methods, namely manual implementation with NumPy and implementation using the PyTorch framework. The main focus of this study is to evaluate the effectiveness and efficiency of both approaches in conducting MNIST classification.

The data used in this study is the MNIST (Modified National Institute of Standards and Technology) dataset. This dataset is obtained through the torchvision.datasets library for use on PyTorch models and then converted into NumPy format for use in NumPy-based manual model training. All data is then normalized by dividing the pixel values into a range of 0 to 1, with the aim of speeding up and stabilizing the model training process.

The neural network model implemented using NumPy is designed as a simple feedforward network with two layers, namely one hidden layer that uses the ReLU activation function and one output layer that uses the Softmax activation function. The entire calculation process, including forward propagation, backward propagation, and weight update, is done manually using numerical operations from the NumPy library. For the optimization process, the Stochastic Gradient Descent (SGD) algorithm is used without the help of other external optimization libraries.

Loss Function – Cross Entropy

Used to measure the difference between true labels y and predictions $\hat{y}$:

$$L = -\sum_{i=1}^{k} y_i \log(\hat{y}_i) \quad (1)$$

Optimization – Stochastic Gradient Descent (SGD)

The weights are updated using:

$$w^{(t+1)} = w^{(t)} - \eta \cdot \frac{\partial L}{\partial w} \quad (2)$$

$$b^{(t+1)} = b^{(t)} - \eta \cdot \frac{\partial L}{\partial b} \quad (3)$$

Softmax Activation + NLLLoss in PyTorch

PyTorch typically combines LogSoftmax with NLLLoss, which is mathematically equivalent to Cross Entropy Loss:

$$\text{LogSoftmax}(z_i) = \log\left(\frac{e^{z_i}}{\sum_j e^{z_j}}\right) \quad (4)$$

$$L = -\log(\widehat{y_{trueclass}}) \quad (5)$$

Principal Component Analysis (PCA)

PCA projects high-dimensional data into a lower-dimensional space (usually 2D):

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \quad (6)$$

$$\Sigma = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})(x_i - \bar{x})^T \quad (7)$$

Binary Classification Function:

To distinguish between two digits (e.g., 3 and 8), a decision rule is used:

$$\hat{y} = \begin{cases} 1, if\ \sigma(z) \geq 0.5 \\ 0, if\ \sigma(z) \geq 0.5 \end{cases} \quad (8)$$

Meanwhile, the PyTorch-based neural network model also uses the Multilayer Perceptron (MLP) architecture which has a similar structure to the NumPy model. The model

is built using the torch.nn library, which consists of an input layer, a hidden layer with a ReLU activation function, and an output layer with a LogSoftmax activation function. The training process was carried out by utilizing torch.optim using the SGD optimization algorithm, while the loss function used was nn. NLLLoss. During the training process, the system records the loss and accuracy values at each epoch for the purpose of evaluation and comparison of performance between the two methods.

Results

4.1 NumPy Model Loss

Based on Figure 4.1, it shows the loss curve of a simple neural network training that is implemented using NumPy. This graph shows the value of the loss on the test data during the 20 epoch of the training. Based on the chart, it can be seen that the initial loss value was above 2.00, then experienced a very sharp decline in the first 3 to 5 epochs. After that, the decline continued more slowly but steadily until it reached a value close to 0.25 in the 20th epoch. A sharp drop at the beginning of the training showed that the model managed to capture the baseline patterns of the MNIST data fairly quickly. This is natural because in the initial phase, the weight of the network is still random and the changes made through the backpropagation process are huge. As the epoch increases, the rate of loss decline slows down as the model begins to make finer adjustments to the weights that are already relatively optimal.

The smaller loss value reflects that the model's prediction errors on the test data are decreasing, indicating that the learning process is going well. In addition, this graph also shows that the NumPy model does not experience significant overfitting symptoms, as the loss value decreases steadily without extreme fluctuations. This shows that even if the model is implemented manually without the help of a deep learning library, the optimization process is still effective in reducing prediction errors. Overall, this curve shows that a simple two-layer neural network built using NumPy is capable of learning MNIST data quite efficiently, and the manually applied Stochastic Gradient Descent-based optimization process successfully delivers stable and convergent results in a relatively short training time.

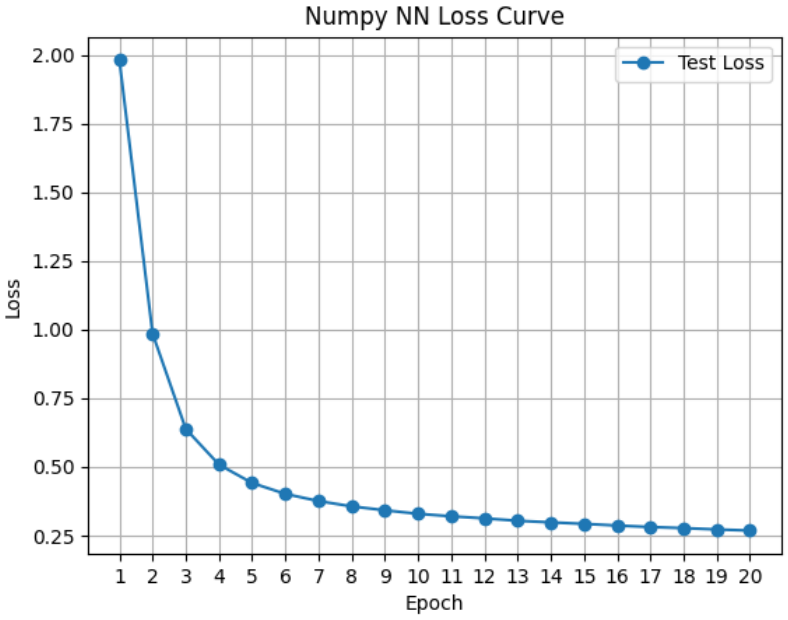

Figure 1. Numpy NN Loss Curve

4.2 Loss Model PyTorch

Figure 4.2 shows the loss curve of a simple neural network training process implemented using PyTorch for 20 epochs. It can be seen that the loss value in the first epoch started from around 1.25, then decreased drastically to around 0.5 in just one epoch. After

that, the downward trend of losses continued gradually and consistently until it reached a value close to 0.20 in the 20th epoch. The shape of this curve shows that the model training process with PyTorch takes place efficiently. The sharp drop at the beginning suggests that the model is able to quickly adjust the weight to the archetypes contained in the MNIST data. This signifies that early optimization is very effective, especially since PyTorch has built-in features such as automatic differentiation and stable numerical optimization.

Not only that, the loss value that continues to decline constantly until the end of the epoch also shows that the model does not experience stagnation (plateau) or overfitting, but continues to make good generalizations on the test data. When compared to the loss curve of the NumPy model, the PyTorch model shows a lower initial loss value and a faster and more stable rate of decline. This is likely due to the use of an integrated optimization library (torch.optim) and a well-calibrated loss function (nn. NLLLoss), both of which are highly optimized for neural network training.

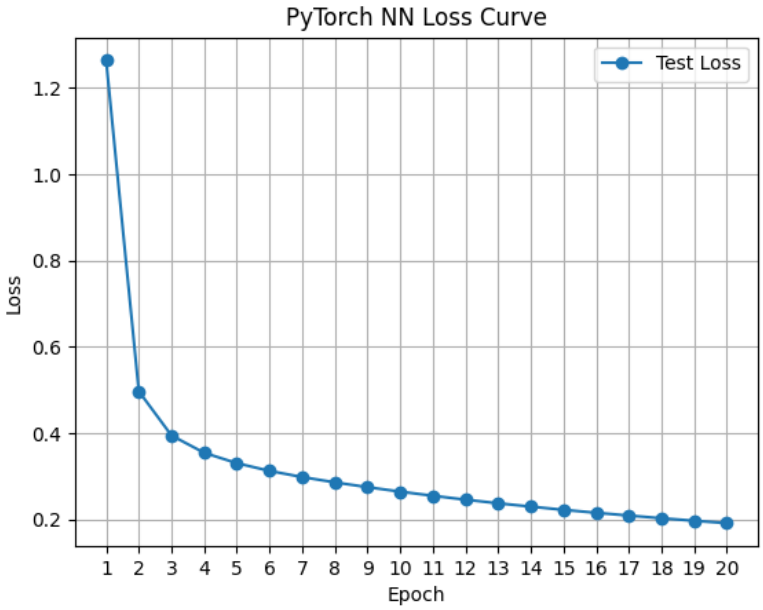

Figure 2. PyTorch NN Loss Curve

4.3 PCA Visualization on MNIST Data

Figure 4.3 shows the results of visualization of MNIST test data using the Principal Component Analysis (PCA) method, which is a dimension reduction technique that aims to project high-dimensional data (784 dimensions on a 28x28-pixel MNIST image) into two main dimensions, namely PC1 (Principal Component 1) and PC2 (Principal Component 2). Each point on the plot represents one data image of numbers, with a different color according to the original digit label from 0 to 9.

From this visualization, it can be seen that several groups of digits begin to form separate clusters even though there is some overlap between classes. For example, the digit '1' (orange) appears to form a fairly isolated cluster on the top left side of the graph, indicating that the pixel pattern of this digit has a fairly strong characteristic. Meanwhile, digits such as '3', '5', and '8' appear to overlap each other, indicating that the visual patterns of the numbers have similarities in their vector representations, making them more difficult to separate with just two main components.

The purpose of the application of this PCA is to visually understand the structure of the data and evaluate whether the features extracted from the numerical images are good enough to distinguish between classes. The results of this PCA show that although the main information is well captured through two components, a perfect classification is still not possible with just two dimensions. However, this is an early indication that machine learning

models such as neural networks still require non-linear features or higher dimensions to be able to perform more accurate classifications.

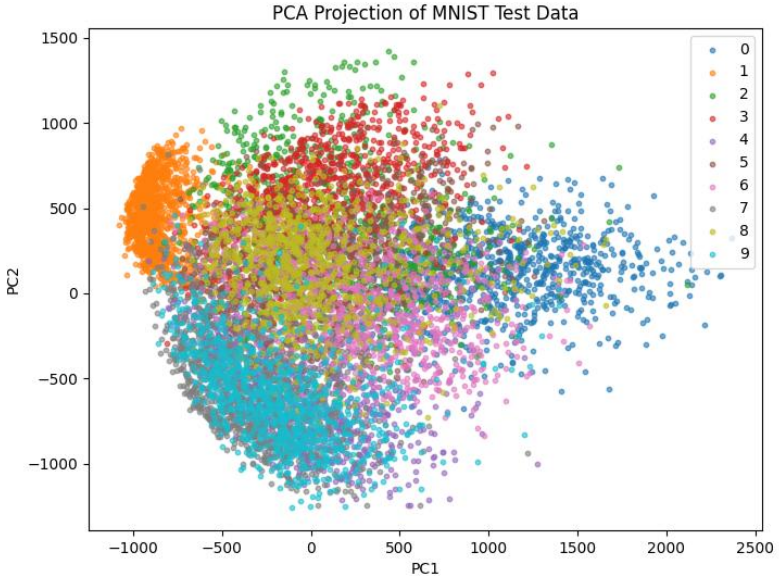

Figure 3. PCA Plot

4.4 Visualization of Decision Boundaries between Digits 3 and 8

Figure 4.4 shows the decision boundary formed by the classification model in distinguishing between two digits that have similar shapes, namely the numbers '3' and '8'. This visualization uses the two main components of the PCA (Principal Component 1 and Principal Component 2) as two-dimensional representations of the original features of the MNIST image data. Each dot in the plot represents one sample image, with blue representing the digit '3' and the red representing the digit '8'. The blue and red backgrounds depict the classification region decided by the model; That is, if a point is in the blue region, then the model classifies it as '3', and vice versa for the red region that shows the prediction as '8'. It appears that the decision boundaries formed are linear and separate the two groups quite well, although there are still some points that are in the wrong territory — this indicates a misclassification or visual similarity between the numbers '3' and '8' that confuses the model. In general, this visualization provides a more concrete understanding of the performance and discriminating capabilities of the classification model against two classes that have a high challenge to distinguish. This visualization of decision boundaries is very useful for visually evaluating the model's performance and for identifying critical areas where the model is prone to error, which can later be the focus for improving the model, for example by adding more complex features or using more powerful algorithms to capture subtle differences between classes.

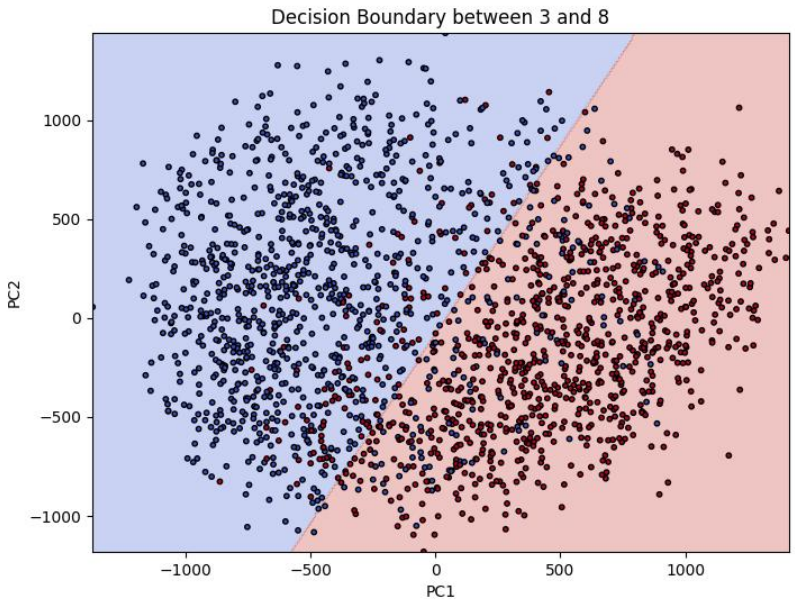

Figure 4. Decision Boundary (3 vs 8)

Conclusion

Based on the results of the research that has been conducted, it can be concluded that neural network training using PyTorch provides superior performance compared to the manual approach using NumPy, both in terms of classification accuracy, training time efficiency, and ease of implementation. The model built with PyTorch is more stable in the training process and is able to produce more defined data distribution visualizations and decision limits, especially in distinguishing similar digits such as numbers 3 and 8. Meanwhile, although training using NumPy provides an in-depth understanding of neural network mechanisms, this approach is less efficient when applied to large scales. The implications of these results show that the choice of framework greatly affects the performance and practicality in the development of machine learning systems. The research also opens up opportunities for further exploration, such as comparisons with other frameworks such as TensorFlow, or testing more complex network architectures to improve the accuracy of hand-digit image classification. In addition, the visualization methods used can be further developed to assist in the interpretation of models in real applications.

References

[1] B. Antunes and D. R. C. Hill, "Reproducibility, energy efficiency and performance of pseudorandom number generators in machine learning: a comparative study of python, numpy, tensorflow, and pytorch implementations," *arXiv preprint arXiv:2401.17345*, 2024.

[2] D. Beohar and A. Rasool, "Handwritten digit recognition of MNIST dataset using deep learning state-of-the-art artificial neural network (ANN) and convolutional neural network (CNN)," in *2021 International Conference on Emerging Smart Computing and Informatics (ESCI)*, 2021, pp. 542–548.

[3] B. Gope, S. Pande, N. Karale, S. Dharmale, and P. Umekar, "Handwritten digits identification using mnist database via machine learning models," *IOP Conference Series: Materials Science and Engineering*, vol. 1022, no. 1, p. 12108, 2021.

[4] I. Gridin, *Automated Deep Learning Using Neural Network Intelligence: Develop and Design PyTorch and TensorFlow Models Using Python*. Springer, 2022.

[5] P. Holl and N. Thuerey, "Φ-ML: Intuitive Scientific Computing with Dimension Types for Jax, PyTorch, TensorFlow & NumPy," *Journal of Open Source Software*, vol. 9, no. 95, p. 6171, 2024.

[6] F. J. J. Joseph, S. Nonsiri, and A. Monsakul, "Keras and TensorFlow: A hands-on experience," in *Advanced deep learning for engineers and scientists: A practical approach*, Springer, 2021, pp. 85–111.

[7] O. Jouini, K. Sethom, A. Namoun, N. Aljohani, M. H. Alanazi, and M. N. Alanazi, "A survey of machine learning in edge computing: Techniques, frameworks, applications, issues, and research directions," *Technologies*, vol. 12, no. 6, p. 81, 2024.

[8] L. Judijanto, A.-A. Al-Amin, and L. Nurhakim, "Implementasi Teknologi Artificial Intelligence dan Machine Learning dalam Praktik Akuntansi dan Audit: Sebuah Revolusi atau Evolusi," *COSMOS: Jurnal Ilmu Pendidikan, Ekonomi Dan Teknologi*, vol. 1, no. 6, pp. 470–483, 2024.

[9] A.-H. M. NA, "Classification and analysis of the MNIST dataset using PCA and SVM algorithms," *Vojnotehnički Glasnik*, vol. 71, no. 2, pp. 221–238, 2023.

[10] O. Nocentini, J. Kim, M. Z. Bashir, and F. Cavallo, "Image classification using multiple convolutional neural networks on the fashion-MNIST dataset," *Sensors*, vol. 22, no. 23, p. 9544, 2022.

[11] O.-C. Novac et al., "Analysis of the application efficiency of TensorFlow and PyTorch in convolutional neural network," *Sensors*, vol. 22, no. 22, p. 8872, 2022.

[12] H. B. P. D. A. Okatan, "Developing a basic neural network to classify images from the MNIST dataset," *Humanities & Natural Sciences Journal*, vol. 4, no. 7, pp. 128–138, 2023.

[13] M. S. Rana, M. H. Kabir, and A. Sobur, "Comparison of the error rates of MNIST datasets using different type of machine learning model," *North Am Acad Res*, vol. 6, no. 5, pp. 173–181, 2023.

[14] I. Hidayat, M. Iqbal, L. Marlina, A. Putera, and U. Siahaan, "Analysis Of Heart Failure Prediction With Random Forest Algorithm And Linear Regression(1)," 2025.

[15] Z. Sitorus, M. Iqbal, D. Nasution, and R. F. Wijaya, "Penerapan Deep Learning dan Analisis Sentimen terhadap Gap Kompetensi Lulusan Lembaga Pendidikan dan Pelatihan Vokasi terhadap Dunia Kerja dengan Metode Long Short-Term Memory (LSTM)," vol. 6, no. 2, pp. 0–11, 2025.