Exploring Deep Learning Techniques for Classification Using PyTorch: MLP on MNIST and CNN on CIFAR-10

*Note: This study was conducted with limited GPU resources; conclusions reflect experimental constraints.

Chonthichar Soythong
Data Science Department
Høyskolen Kristiania
Oslo, Norway
Email: miso025@student.kristiania.no

***Abstract*—This report investigates how deep learning models handle classification tasks on two datasets, MNIST and CIFAR-10. We utilize Multi-Layer Perceptrons (MLP) for MNIST and Convolutional Neural Networks (CNN) for CIFAR-10, experimenting with three optimizers—SGD, ADAM, and RMSProp—alongside learning rate schedulers (ReduceLROnPlateau, StepLR, and ExponentialLR) and data augmentation techniques. The aim is to analyze how these combinations affect training and testing accuracy, loss, and generalization. The CNN model achieved the accuracy ranging from 10-93% on CIFAR-10, while the MLP model reached 60-99% on MNIST. The results highlight the efficacy of deep learning, though future work could explore improvements through optimized architectures and additional techniques.**

***Index Terms*—Deep Learning, Data Augmentation, PyTorch, Classification, Optimizers, MNIST, CIFAR-10**

I. Introduction

In image classification, distinguishing between objects such as dogs and horses, or handwritten digits is a common task in computer vision. This study focuses on developing the model, making it capable of identifying images using deep learning techniques, and implementing two popular datasets, CIFAR-10 and MNIST, to understand their performance in image recognition and digit classification. To further enhance model performance, we employed data augmentation techniques, optimizers, and learning rate schedulers, which allows us to explore how each combination influences model accuracy, generalization, and robustness. Using PyTorch and tensor operation as tools to handle the tasks. The report begins with an overview of relevant exploring, followed by a detailed description of our methodology and results.

II. Methodology

The methodology will follow a logical progression consistent with the workflow depicted in Figure 1:

This study was conducted at [Høyskolen Kristiania].

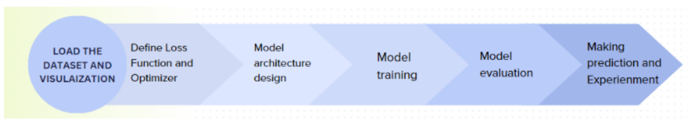
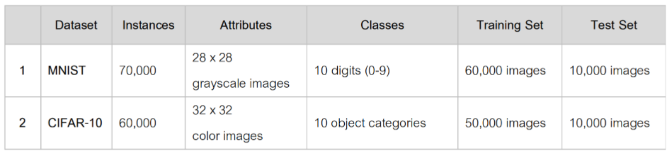

LOAD THE DATASET AND VISUALIZATION
Define Loss Function and Optimizer
Model architecture design
Model training
Model evaluation
Making prediction and Experienment

Fig. 1. Data Pipeline.

A. *Prepare the Dataset*

a) : Load the dataset

In this study, we use PyTorch to classify images, specifically using a Convolutional Neural Network (CNN) for CIFAR-10 and a Multi-Layer Perceptron (MLP) for MNIST. Both datasets were downloaded from torchvision.datasets module [9], as shown in Figure 2.



	Dataset	Instances	Attributes	Classes	Training Set	Test Set
1	MNIST	70,000	28 x 28 grayscale images	10 digits (0-9)	60,000 images	10,000 images
2	CIFAR-10	60,000	32 x 32 color images	10 object categories	50,000 images	10,000 images

Fig. 2. MNIST and CIFAR-10 Dataset Characteristics

b) : Define Transformations

Data Preprocessing step: To prepare the data, we applied transformations including transforming images to tensor format and normalization to standardize pixel value. [5]

Data Augmentation: We applied geometric transformations as data augmentation techniques across both datasets, including rotating images, horizontal flipping, and cropping. According to studies, using a range of transformations, helps improve generalization in unknown data and can significantly increase model accuracy by decreasing overfitting and creating a more diverse dataset. [5]

c) : Create Data Loader

After defining transformations, **a data loade**r is set up to splits the dataset into batches, each batch containing a set of images and corresponding labels. The dataset is first split into training and test sets, and a specified batch size is defined before being passed into the DataLoader. **Bath size** refers to the number to show how many images will go through the network before we train and update.

In short, the data set is downloaded, then transformation (including augmentation) used in each batch to transform every image, and the data loader will then be able to send these batches to the model during training.

d) : Data Exploration

Before training, we explore the dataset to understand the class balance and ensure it has been loaded correctly.

MNIST: We examined class distribution by plotting the frequency of each digit (0-9) in the dataset using Matplotlib. We then verified the size of both image and label batches, followed by printing the labels of the first few images in each batch to ensure correct mapping between the images and labels (Figure 1).

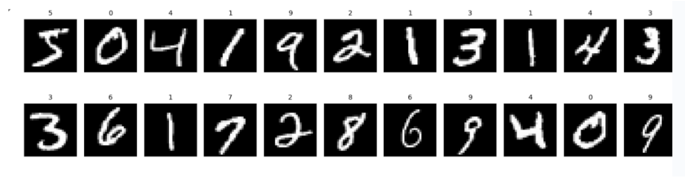


Fig. 3. Displays multiple images from the MNIST dataset along with their corresponding labels.

CNN: Similarly, we visualized a subset of the images as shown in Figure 4. Each image in the dataset is labeled with its respective class; **Airplane, Automobile, Bird, Cat, Deer, Dog, Frog, Horse, Ship and Truck.**

Fig. 4. Displays A grid of sample images from the CIFAR-10 dataset

Through this exploration, we obtained a comprehensive view of the dataset, helping to determine the most suitable model architecture for each dataset. Finally, all training processes were performed in a GPU-enabled environment to accelerate computation and improve performance.

B. Define the Model Architecture

Finally, with all the necessary building functions, it is time to define the model. The model covered in this report is MLP and CNN, these model is widely used and prove the ability to analyze images classification and make accurate predictions, providing a solid foundation on our research on finding the most accuracy model:

a) : Multi-Layer Perception (MLP)

The MLP is a feed-forward neural network consisting of multiple fully connected dense layers with a nonlinear activation function and able to distinguish data that is not linearly separable. [3][4]

1) **Input Layer:** The input layer of an MLP gets the dataset's features and passes them to the hidden layers.
2) **Hidden layers:** Neurons compute outputs using weighted inputs and activation functions to capture complicated patterns.
3) **Output layer:** the last dense layer is output layer which contains 10 neurons the final output is typically passed through a softmax function for multi-class classification.

MLP for MNIST: We developed 1,2 hidden layers of 512 neurons each. The input Layer is a flattened 28 x 28 into a vector of 784 features, Hidden layer - has two hidden layers, each with 512 and the layer output consists of 10 neurons, corresponding to the 10 digits classes (e.g., digits 0-9 for MNIST) then dropout (20%) after each hidden layer to prevent over-fitting.

b) : Convolutional Neural Networks (CNN)

CNNs are deep learning architectures designed to automatically learn features from data, particularly images. They consist of three main types of layers:

1) **Convolutional layer:** Apply filters (small grids) to the input images. The filter slide across the image and detect features like edges, patterns or textures. [7]
2) **Pooling layer:** Reducing the dimensionality of the data for example max or average pooling until the data is small enough for fully connected layers to classify the image. [7]
3) **Fully connected Layers:** These layers take all the learned features and combine them to classify the image or make the prediction. [8]

CNN for CIFAR-10: The CNN model includes three convolutional layers, each followed by ReLU activation and max pooling:

- Conv Layer 1: 32 filters, 3x3 kernel, ReLU, MaxPooling (2x2)
- Conv Layer 2: 64 filters, 3x3 kernel, ReLU, MaxPooling (2x2)
- Conv Layer 3: 128 filters, 3x3 kernel, ReLU, MaxPooling (2x2)

After the final convolutional layer, the output is flattened into a 1D vector and passed through fully connected layers (256, 128 neurons) with dropout (50%) to prevent overfitting. The final layer outputs 10 neurons with softmax activation.

C. Define the Loss Function, Optimizer and Learning rate scheduler

In Deep Learning with Python, Ketkar and Moolayil (2021) discuss several techniques of optimization algorithms and learning rate schedulers, which are important in fine-tuning model parametrs and can improve model performance. The following techniques are implemented in this report:

- **Loss Function:** The cross Entropy Loss compares the predicted probability value (between 0 and 1) with the true label to calculate the loss which is often used in classification tasks.
- **Optimizers:** Optimizers are algorithms used to update the model's parameters (weight and biases) to reduce the loss, allowing the model to improve over time. Three optimizers were tested:
  1) **SGD** (Stochastic Gradient Descent): SGD updates the model weights by calculating the gradient of the loss for each training example.
  2) **Adam** (Adam gradient descent): Adam algorithm optimizer computes the gradient, then maintains the moving averages of gradients and squared gradients, adjusts the learning rate, and updates the weights.
  3) **RMSprop** (Root Mean Squared Propagation): RMSprop initializes state variables to store the moving average of squared gradients, scale gradients, and update the model parameters using the scaled gradients and learning rate.
- **Learning rate scheduler:** control the learning rate changes during training, helping the model learn effectively:
  1) **StepLP:** is a scheduling technique that reduces the learning rate by a fixed factor (gamma) number of every N epoch.
  2) **ReduceLROnPlateau:** decrease the learning rate when the specified metrics such as validation loss stop improving for a defined number of epochs (patience). [5]
  3) **ExponenetialLR:** decays the learning rate smoothly over time. [5]
- **Epoch:** Define the number of epochs or parameters that delicate the number of times that the learning algorithm will work through the entire training dataset, each time updating the weights based on the loss after the epoch pass.

In this study, we used Cross Entropy Loss for both datasets with three optimizers SGD, Adam, and RMSProp for our network. The model was trained for 20 times (epochs) for MNIST and 30 times (epochs) for CIFAR-10 with the following specific learning rate scheduler of 0.001; StepLR, ReduceLROnPlateau and ExponentialLR learning rate Scheduler.

D. Model training

After all functions are in place, the model is then trained using PyTorch's model. train() mode, where each batch of the data from the training dataset goes through the following steps:

1) **Forward Pass:** Pass the batch of images through the model to generate predictions.
2) **Loss Calculation:** Uses Cross-Entropy Loss to compare predictions with true labels.
3) **Backpropagation:** The function computes the gradients of the loss.
4) **Weight Adjustment**: Weights are updated with optimizer.step() in the direction that minimizes loss.
5) **Learning rate scheduler adjustment:** Adjust the learning rate according to the specific scheduler.
6) **Logging Metrics:** Training and test loss, accuracy are recorded.

In summary, the model is trained over a specified number of epochs, where it performs forward passes, calculates loss, adjusts weights using backpropagation and the optimizer, and applies the learning rate scheduler. Finally, metrics are recorded throughout the training process to monitor the model's performance effectively.

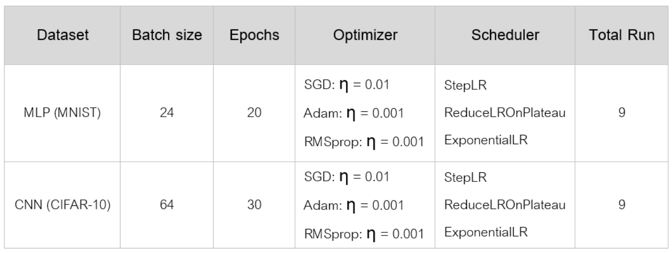
E. Model evaluation

We set the model to evaluation mode by calling model.eval(), which disables gradient computation. Because both tasks are classification problems, we compute performance metrics, including accuracy by comparing predicted labels with the true labels, and measuring the proportion of correctly classified images out of the total images.**The average test loss** measures how well the model's predictions align with the true labels on unseen data, while the test accuracy quantifies the percentage of correct predictions made by the model. The test data is evaluated, with Cross-Entropy Loss calculated for each batch. **The average test loss and test accuracy were recorded**, with final values of 20 epoch (MNIST) and 30 epoch (CIFAR-10) considered the final results of the experiment.

F. Making Prediction and Experiment

Next, we conducted several experiments to identify the best configuration :

1) **Testing optimizer and learning rate scheduler:** We conducted a total of 18 experiments (9 for MLP on MNIST, 9 for CNN on CIFAR-10), combining three optimizers and three learning rate schedulers. The results were compared based on training and test accuracy/loss.



Dataset	Batch size	Epochs	Optimizer	Scheduler	Total Run
MLP (MNIST)	24	20	SGD: η = 0.01 Adam: η = 0.001 RMSprop: η = 0.001	StepLR ReduceLROnPlateau ExponentialLR	9
CNN (CIFAR-10)	64	30	SGD: η = 0.01 Adam: η = 0.001 RMSprop: η = 0.001	StepLR ReduceLROnPlateau ExponentialLR	9

Fig. 5. Experiment Configurations for MLP (MNIST) and CNN (CIFAR-10)

2) **Testing data augmentation:** We also ran an experiment to test three different data augmentation techniques with one of our models; Experiment 2 - Adam with StepLR scheduler on MNIST, and Adam with ReduceLROnPlateau on CIFAR-10 which showed poor generalization

and high overfitting in both dataset. The data augmentation techniques tested included rotation, horizontal flipping, and cropping.

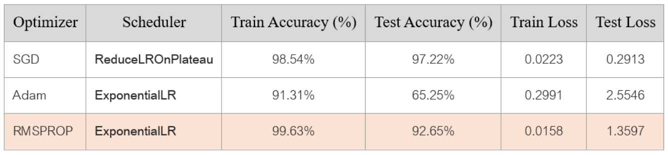
We use a single training model and evaluation functions across the experiments. This helps to reduce the redundancy in coding during the execution of each experiment and minimizes the risk of memory overload in deep learning environments.

III. Analysis and results

From 18 tuning experiments, we selected the best-performing network for each optimizer, as shown in the tables and graphs below.

A. MNIST with MLP model

Best Optimizer and learning rate scheduler techniques:



Optimizer	Scheduler	Train Accuracy (%)	Test Accuracy (%)	Train Loss	Test Loss
SGD	ReduceLROnPlateau	98.54%	97.22%	0.0223	0.2913
Adam	ExponentialLR	91.31%	65.25%	0.2991	2.5546
RMSPROP	ExponentialLR	99.63%	92.65%	0.0158	1.3597

Fig. 6. Best Optimizer and Scheduler Performance Metrics

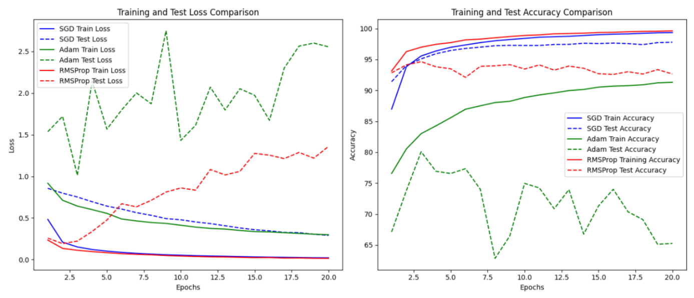


Fig. 7. MNIST - Training and Test Metrics Comparison for Optimizers and Schedulers

- **Experiment 1 (SGD with ReduceLR scheduler)**:
In the table, the highest accuracy (98.54%) was achieved with the ReduceLR scheduler. **The blue plot line** shows a balanced accuracy increase for both training and test set, indicating good generalization as seen in the small gap with minimal over-fitting. Train and test losses were stable and low, making it the best performance among other SGD configuration
- **Experiment 2 (Adam with ExponentialLR):**
Adam with ExponentialLR reached the best result (91.31%). However, None of the Adam configurations performed well in terms of generalization as it showed high over-fitting. As seen in **the green plot line**, the test loss shows high variance and spikes, while the training accuracy remains high, suggesting the model is memorizing the training data without effectively generalizing to new data.
- **Experiment 3 (RMSProp with ExponentialLR):**
RMSProp shows the highest accuracy across the overall experiment (99.63%) using ExponentialLR. **The red plot line** shows that RMSprop had the lowest and smoothest training loss, suggesting it learns the patterns effectively. However, the test loss shows some inflection, indicating the moderate over-fitting.

Data augmentation technique: Adam and StepLR learning rate scheduler were selected to evaluate the data augmentation technique because of its poor initial metric performance that performed moderately with results between 88-91% , poor generalization and high over-fitting. From applying rotation, horizontal flipping and cropping improved accuracy to around 94%, with better results than without augmentation. However, training for 20 epochs led to under-fitting, where test accuracy was initially higher than train accuracy, but with more epochs, training accuracy surpassed the test accuracy. The result proved that data augmentation improved overall model performance, and extending training further could help the model learn better.

Random Validation on MNIST: We performed random validation on 10 samples from the MNIST test set to visually inspect the model's predictions alongside the true labels. This visualization provides insight into the model's performance with the **SGD optimizer and StepLR scheduler**, achieving a test accuracy of **97.49%** after 20 epochs with stability and no signs of over-fitting. The image shows cases where the model achieved 100% accuracy on the selected samples, showing its effectiveness in classification tasks.

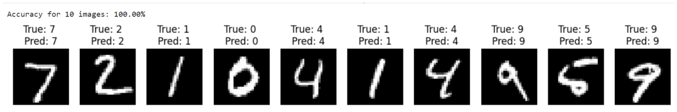


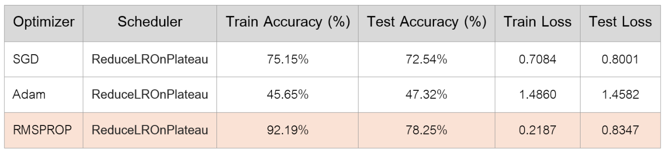
Fig. 8. Random validation on MNIST with MLP Model (SGD with StepLR scheduler)

In summary, all optimizers achieved high accuracy scores between 70% and 99%. The MLP model provides the highest accuracy with **RMSP and ExponentialLR** - shows the best performance, however we observed some slight overfitting. **SGD with ReduceLR** achieved a slightly lower accuracy score, but showed the best generalization, with a balanced performance between training and test accuracy and providing the most suitable model for MNIST. In contrast, **Adam** struggled with consistent test accuracy, showing high variance and over-fitting. **Data augmentation technique** improved the model's generalization on unseen data compared to training without augmentation.

B. CIFAR-10 with CNN

Best Optimizer and learning rate scheduler techniques:

- **Experience 1 (SGD with ReduceLROnPlateau):** gave a final training accuracy at 66.84%, **blue lines** improved gradually, reaching a good number of test accuracy at the end. The train test accuracy gap is small, showing a good generalization but slower learning curve. While



Optimizer	Scheduler	Train Accuracy (%)	Test Accuracy (%)	Train Loss	Test Loss
SGD	ReduceLROnPlateau	75.15%	72.54%	0.7084	0.8001
Adam	ReduceLROnPlateau	45.65%	47.32%	1.4860	1.4582
RMSPROP	ReduceLROnPlateau	92.19%	78.25%	0.2187	0.8347

Fig. 9. Best Optimizer and Scheduler Performance Metrics

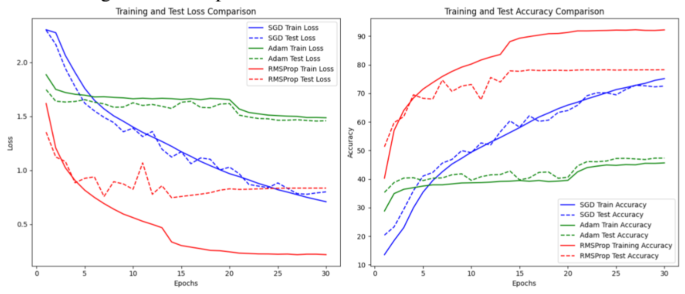


Fig. 10. CIFAR-10 - Training and Test Metrics Comparison for Optimizers and Schedulers

both training and test loss show steady decreases . But by the end of epochs, SGD test loss remains higher than RMS indicating slightly less effective generalization.

- **Experience 2 (Adam with ReduceLROnPlateau):** Reached the final train accuracy at 47% but showed small improvement overtime(**green line**). Both train and test loss accuracy remains high, with only slightly decline. This suggests that Adam struggles with this setup and may not be effectively learning patterns in this data just like theory of two optimizer the Step and Exponential that only reach around 10% accuracy after 30 epochs.
- **Experience 3 (RMSProp with ReduceLROnPlateau): red lines** reach the highest training and test accuracy among all optimizers, the test accuracy gap moderately below the training accuracy, showing that RMSP performs well on unseen data with slightly over-lifting. While in train and test loss archives the lowest training and test overall, remaining a low loss through the training, showed strong generalization with small over fitting.

Data augmentation technique: Using augmentation techniques (Rotation, Horizontal Flipping and Cropping) with Adam optimizer and ReduceROn Plateau scheduler improved test accuracy and reduced losses, showing improvement in generalization. While, notice that the gap between training and test accuracy narrowed, suggesting improved stability, but further tuning could improve the performance.

Random Validation on CIFAR-10: Similarly, we conducted random validation on 10 samples from the CIFAR-10 test set using the same SGD optimizer with StepLR scheduler. The model achieved a test accuracy score of 56.04% after 30 epochs. This validation highlights specific cases where the model succeeds and struggles in classification.

In summary, the CNN model performed differently de-

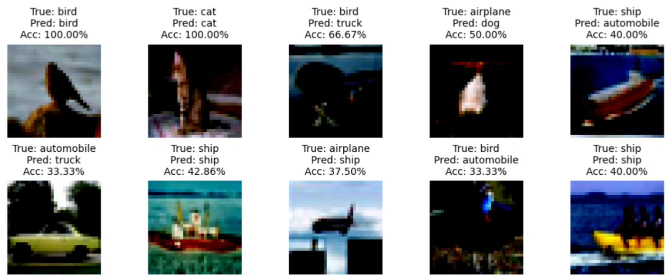


Fig. 11. Random validation on CIFAR-10 with CNN Model (SGD with StepLR scheduler)

pending on the optimizer. **RMSProp with Reduces learning rate scheduler** shown proved to be the most effective optimizer, providing high accuracy and stability with minimal over-fitting, followed by **StepLR optimizer with ReduceLROnPlateau** which achieve reasonable results. Notably, all optimizer performed best with the ReduceLROnPlateau, showcasing the importance of selecting an appropriate learning rate scheduler. However, **Adam** struggles with both loss and accuracy, suggesting issues with its compatibility with the dataset and model configuration. Given the low accuracy and fluctuations observed from CNN architecture used, a more complex model may better suit for CIFAR-10.

IV. Discussion and Improvements

- **Model Selection:** The performance of train and test loss/accuracy on CIFAR-10 shows that utilizing simple CNN architecture for complex datasets like CIFAR-10 may not be most suitable. It recommends using modern architecture such as efficient adaptive ensembling, to better address the complexity of such datasets more effectively.[9] However, because of hardware limitations (e.g., lack of advanced GPUs), testing a more complex model was not feasible in this study.

- **Data Augmentation:** Data augmentation techniques were applied in this dataset, however it is important to be aware that their effectiveness may vary based on the type of images. For example, flipping may not always be appreciated for datasets like CIFAR-10, where the orientation of animals matters. [8]

- **Batch Size Impact:** In this study proves that the batch size was found to have a significant effect on model performance. A larger batch size (128) in CIFAR-10 led to lower accuracy, while reducing the batch size to (64) improved the model's convergence. Research by Facebook AI [6], suggests that large batch sizes can speed up training but require careful tuning of the learning rate to maintain accuracy.

- **Epochs:** On CIFAR-10, the model initially trained for 20 epochs, but this didn't give satisfactory accuracy. Therefore, the training was extended to 30 epochs, which resulted in some improvement. Similar with MNIST dataset, when using data augmentation technique into the Adam and StepLR scheduler. Initially, test accuracy scores are higher than train accuracy score, which is common in the early stages of training when the model hasn't fit the training data well yet. This is due to underlifting, where the model is still learning general features and hasn't yet captured the specific patterns. Increasing the number of epochs could improve the model's performance by allowing more time to learn from the data.[5] For example, CIFAR-10; experimented with SGD and Reducer scheduler, training accuracy eventually suppresses test scores after 20 epochs.

- **Overfitting Mitigation:** In this report, we found high overfitting in some experiments, particular with Adam optimizer. This finding suggested that techniques like early stopping or cross validation that's set earlier in the training process could address over-fitting and improve generalization to unseen data. [5]

V. Conclusion

In this study, we performed different experiments to examine insights of the model's metric performance, generalization and the impact of different optimizers, learning rate schedulers and data augmentation techniques ON the MNIST and CIFAR-10 datasets. Notably, **The RMSProp optimizer, especially when paired with the ReduceLROnPlateau learning rate scheduler**, produced the best results, showing high accuracy and consistent learning across both datasets. Following closely, **the SGD optimizer with the ReduceLROnPlateau scheduler** also provided stable and balanced performance. While applying data augmentation slightly improved the model's ability to handle new data, its impact on CIFAR-10 was less clear. Further improvement may be achieved by adjusting the model configuration for optimal results.

References

[1] CIFAR-10 and CIFAR-100 datasets. Accessed: Oct. 17, 2024. [Online]. Available: https://www.cs.toronto.edu/ kriz/cifar.html

[2] Wikipedia contributors, "Multilayer perceptron," *Wikipedia*, Aug. 7, 2024. [Online]. Available: https://en.wikipedia.org/wiki/Multilayer$_perceptron$

[3] GeeksforGeeks, "MultiLayer Perceptron Learning in Tensorflow," Nov. 5, 2021. [Online]. Available: https://www.geeksforgeeks.org/multi-layer-perceptron-learning-in-tensorflow/

[4] K. Upadhyay, "Learning Rate Schedulers - Karunesh Upadhyay - Medium," *Medium*, Jun. 30, 2024. [Online]. Available: https://medium.com/@karuneshu21/learning-rate-schedulers-9fc8c05f0019

[5] CloudFactory, "Comprehensive overview of loss functions in Machine Learning." [Online]. Available: https://wiki.cloudfactory.com/docs/mp-wiki/loss/comprehensive-overview-of-loss-functions-in-machine-learning

[6] P. Goyal et al., "Accurate, large minibatch SGD: Training ImageNet in 1 hour," *arXiv preprint arXiv:1706.02677*, 2017. [Online]. Available: https://arxiv.org/abs/1706.02677

[7] N. Ketkar and J. Moolayil, *Deep Learning with Python*. Springer, 2021. [Online]. Available: https://doi.org/10.1007/978-1-4842-5364-9$_2$

[8] Papers with Code, "State-of-the-art Image Classification on CIFAR-10." [Online]. Available: https://paperswithcode.com/sota/image-classification-on-cifar-10

[9] L. Prechelt, "Automatic early stopping using cross-validation: Quantifying the criteria," *Neural Networks*, vol. 11, no. 4, pp. 761-767, 1998. [Online]. Available: https://doi.org/10.1016/S0893-6080(98)00010-0

[10] *Programming PyTorch for Deep Learning*. O'Reilly Online Learning. Accessed: Oct. 17, 2024. [Online]. Available: https://learning.oreilly.com/library/view/programming-pytorch-for/9781492045342/ch02.htmlpytorch-and-dataloaders