PGR207 Deep Learning

* Mandatory Assignment

Abstract – in this experiment I explore the performance of deep learning models on the MINST and CIFAR-10 datasets using different optimization techniques, learning rate schedulers, and data augmentation methods. A Multi-layer Perception (MLP) has been implemented for the MINST dataset, and a Convolutional neural Network (CNN) for CIFAR-10. Both have been implemented with the use of PyTorch. The results show different optimization methods and learning rate schedules significantly impact model accuracy and training efficiency. This report presents a detailed analysis and comparison of these methods.

I. Introduction

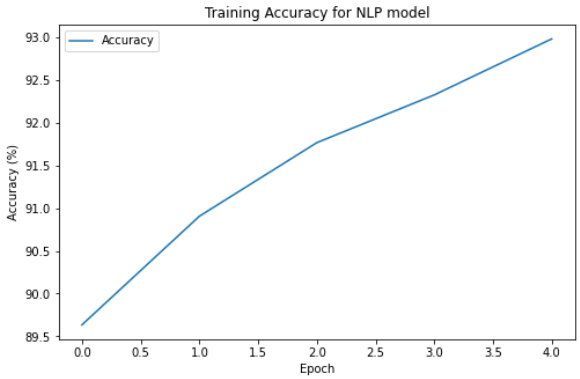
The growing importance of deep learning in computer vision tasks necessitates exploring how model configurations affect performance. This report examines the impact of three data augmentation techniques, three learning rate schedulers, and three optimization methods both on MINST and CIFAR-10 datasets. The main aim is to identify configuration that maximizes accuracy and efficiency in image classification tasks.

II. Model Architecture

The MINST dataset consists of collection of handwritten digits from 0 to 9. There are 28 grayscale images displayed in this library, each measuring 28x28 pixels. While the CIFAR-10 dataset contains 60000 32x32 color images arranged in 10 classes (e.g., Trucks, cats, ships, etc.), each with 6000 images. There are five training batches and one test batch, each with 10000 images. The test batch contains 1000 randomly selected images from each class. The training batches contain 5000 images form each class [1].

I started first with implementing the neural networks, MLP for MINST and CNN for CIFAR-10. For the MINST dataset I implemented MLP with one single hidden layer with 128 neurons with activation function, ReLU. And the output layer has 10 neurons, which corresponds to the 10 classes in the MINST dataset. The reason for that is because I believe that a single hidden layer with only 128 neurons is typically sufficient for a simple dataset with 28x28 grayscale images of handwritten digits.

On MLP I trained a model to run 5 epochs showing the loss functions and accuracy. It means that the given loop, num_epochs, is set to 5 and will train for 5 complete passes over the training data. I implemented the loss function to measure how well the model's predictions align with the actual labels. It will quantify the difference between the predicted values and the true values. I expressed the accuracy as a percentage in order to be easy to interpret and understand when measuring the portion of correctly predicted labels.



The image shows us the final performance of the training we implemented for NLP on MINST dataset. We see that model's accuracy progression over the training epochs increases just only with one hidden layer with 128 neurons. There was no reason for me to increase or add another hidden layer for that reason. The architecture is a good balance between simplicity and effectiveness, yielding an accuracy of around 86-93% after training

With CIFAR-10, I went for three convolutional layers with ReLU activation where the number of channels is increasing from 32 to 128 in order to fit more complex patterns. This type of activation function is used throughout for helping to speed up training even though it took longer time to get an output than MINST dataset. Max pooling follows each convolution, leading to reduced dimensions and computational load. The 256 and 128 neurons, fully connected layers process the high-level features to map on the final output classes, which is 10. This type of design balances between depth and efficiency, enabling effective image classification for CIFAR-10 while maintaining computational cost.

III. Data Augmentation Techniques

For the MINST dataset I applied three different data augmentation to the same random image to enhance model generalization. The variability is introduced by the horizontal flipping by mirroring the digit even though it might not be right to apply for MINST dataset. It can add slight complexity and robustness. Same for the rotation transformation. This type of transformation rotates digits as I prefer, and in this case, I used *degrees=(60,90),* stimulating different writing angles and helps recognizing numbers written with tilts. Affine transformation

is a simple operation that scales, shears, or translates the image – similar to what would happen naturally with variations in handwriting. That was particularly the reason I chose to use to add realistic distortions to the dataset.

I used color jitter, horizontal flip and random crop to expand the model generalizing capacity. The color jitter separately adjusts brightness, contrast, saturation, and hue which makes the model much more robust to variations in color that we can find between different images. The same concept follows when it comes to horizontal flip. With random crop, the images resize and randomly crops few parts of the images. It will improve the model's ability to focus on different parts of an image.

IIII. Optimization Methods

For MINST dataset I used Stochastic Graident Decent (SDG), Adaptive Moment Estimation (Adam Optimizer), and Root Mean Squared Propagation (RMSProp) as optimization. These optimization algorithms are very popular. SDG is an iterative optimization process where it reaches an objective function's minimum and maximum value, and it is the most used methods for changing a model's parameters to reduce a cost function in ML projects [4]. After training the model to use the optimizer with only 5 epoch each, we got expected results.

```
-- Epoch [1/5], Loss: 0.8732, Accuracy: 80.56%
-- Epoch [2/5], Loss: 0.3752, Accuracy: 89.68%
-- Epoch [3/5], Loss: 0.3200, Accuracy: 91.05%
-- Epoch [4/5], Loss: 0.2893, Accuracy: 91.84%
-- Epoch [5/5], Loss: 0.2658, Accuracy: 92.49%
```

The image above shows the result of SDG. We see that the accuracy increases from 81% to 92%. Here the train model, train_model, function accepts the model, loss function and the optimizer, and then iterates through the training data, train_dl, in order to update the model weights and show the performance. The structure of the train_model function remains the same for other optimizers. However, compared to SDG, The Adam optimizer has a few pros: it is a technique implementing adaptive learning rate to improve training speeds and reach the results quickly [5]. These generally result in faster and more stable convergence. We can see the accuracy result here differs from the result I got from SGD. The accuracy went from 92% to 98%.

```
-- Epoch [1/5], Loss: 0.2941, Accuracy: 91.88%
-- Epoch [2/5], Loss: 0.1279, Accuracy: 96.28%
-- Epoch [3/5], Loss: 0.0869, Accuracy: 97.39%
-- Epoch [4/5], Loss: 0.0658, Accuracy: 97.96%
-- Epoch [5/5], Loss: 0.0508, Accuracy: 98.36%
```

The adaptive learning model, RMSProp, is designed to improve the performance and speed of training deep learning models [6]. RMSProp gave us almost the same result as Adam optimizer. Both optimizers have their own strengths but different ways to handle the task. Adam optimizer combines two techniques, momentum to accelerate and adaptive learning rates [7]. RMSProp is more straightforward but the gave the same result as Adam Optimizer.

With CIFAR-10 I used AdamW, Adagard and AdamX as the optimations. AdamW optimization, we use adaptive estimation of both first and second order moments, as well as an added method oof decaying weights. These techniques are discussed Loshchilov, Hutter and colleagues in the paper, "Decoupled Weight Decay Regularization" [8]. The infinity norm variant of Adam, Adamax is a first-order gradient-based optimization method. It has the capability of adjusting the learning rate based on data characteristics and it is well suited to learn time-variant process. That is why we can say that Adamx is useful for high dimensional data [9]. AdaGrad is a family of sub-gradient algorithms for stochastic optimization and scales learning rates base on the sum of squared gradients, causing diminishing rates over time. And it is well suited for sparse data [10]. In my opinion, optimizers that are capable of handling different types of challengers in training will benefit from the complex and diverse image data in the CIFAR-10 dataset. AdamX is well-suited for high dimensional data, making it useful in deep networks with numerous parameters that must be updated to be reliable. Adagard optimizes modified learning rates learning rates for each parameter independently, which can be useful for the CIFAR-10 dataset since various characteristics inside the images may converge at different rates and increase flexibility. AdamW on the other hand, will help with improving the generalization. I put those on the test to see what result it would bring. AdamW performance shows a substantial improvement on accuracy. It reaches 82% which can be assumed as effective learning. With Adagrad optimizer did the work slowly compared to AdamW. It reached 35%. It shows that it struggles to reduce loss effectively. The AdamX optimizers accuracy increased steadily, starting from 36% to 47%. It shows that it is a improvement over Adagrads performance and lower than AdamW. Overall, AdamX be an effective optimizer, but AdamW is most likely to be a more suitable option for higher accuracy on CIFAR-10.

```
Training with Adamax Optimizer
Epoch [1/5], Loss: 1.7403, Accuracy: 36.11%
Epoch [2/5], Loss: 1.6258, Accuracy: 40.29%
Epoch [3/5], Loss: 1.5758, Accuracy: 41.91%
Epoch [4/5], Loss: 1.5225, Accuracy: 44.46%
Epoch [5/5], Loss: 1.4637, Accuracy: 46.56%
```

```
Training with Adagrad Optimizer
Epoch [1/5], Loss: 2.0815, Accuracy: 22.23%
Epoch [2/5], Loss: 1.9326, Accuracy: 28.68%
Epoch [3/5], Loss: 1.8588, Accuracy: 31.57%
Epoch [4/5], Loss: 1.7995, Accuracy: 33.70%
Epoch [5/5], Loss: 1.7602, Accuracy: 35.42%

Training with AdamW Optimizer
Epoch [1/5], Loss: 1.4062, Accuracy: 48.53%
Epoch [2/5], Loss: 0.9514, Accuracy: 66.28%
Epoch [3/5], Loss: 0.7576, Accuracy: 73.57%
Epoch [4/5], Loss: 0.6189, Accuracy: 78.40%
Epoch [5/5], Loss: 0.5094, Accuracy: 82.30%
```

The epoch results show both Loss, Learning Rate (LR), and Accuracy. The Loss Function decreases rapidly from 0.8648 in the first epoch all the way to 0.2459. This decrease shows that the model is learning and minimizing the error. We see the same process on the learning rate as well. The rate started at 0.01000 and dropped to 0.000010. But on the Accuracy, the accuracy improved quickly. It reached up to 92% by epoch 5 and continued to increase up to 93 by epoch 15.

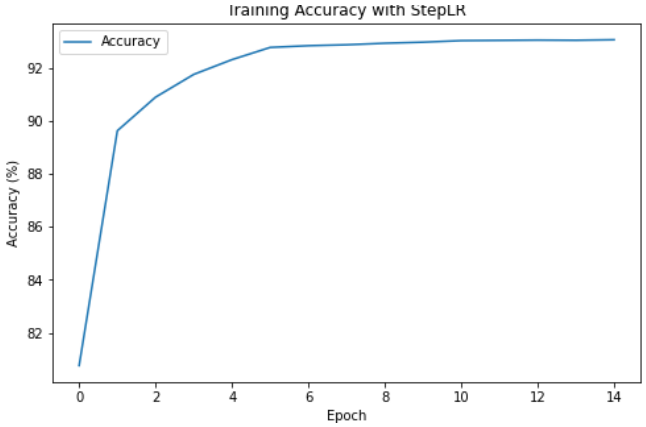
The results with StepLR show us the model is converging and the learning rate model is helping with the improvement and does the job as it describes.

ExponentialLR reduces the learning rate since it divides the learning rate after each epoch, progressively lowering it over time by the same factor called gamma [3].

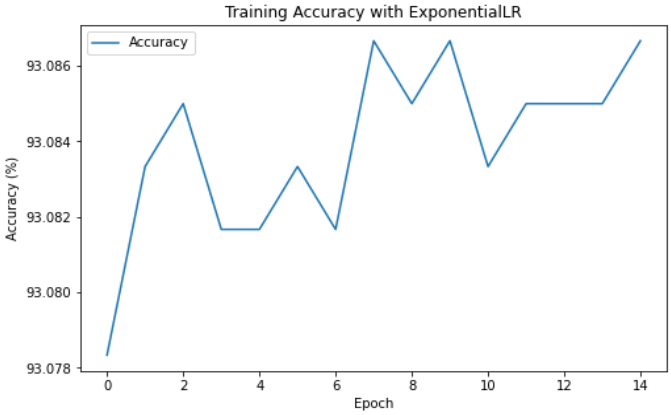
IIIII. Learning Rate Methods

Optimized performance on the training process and the improvement model process will show us good results when we use different learning rate schedules. I thought it was worth to try Exponential Learning Rate Scheduler (ExponentialLR), Reduce Learnin Raye on Plateau (ReduceLROnPlateau) and Step Learning Rate Scheduler (StepLR) on MINST dataset. StepLR decreases the learning rate by gamma every given epoch periods [2]. The learning rate model improved well quickly but at the end of the epoch run it needed to take smaller steps to fine-tune for reaching higher accuracy. It was what I hoped for before running the training. I trained the model with 15 epoch to see the longer process:

```
-- Epoch [1/15], Loss: 0.8648, LR: 0.010000, Accuracy: 80.76%
-- Epoch [2/15], Loss: 0.3736, LR: 0.010000, Accuracy: 89.63%
-- Epoch [3/15], Loss: 0.3206, LR: 0.010000, Accuracy: 90.91%
-- Epoch [4/15], Loss: 0.2909, LR: 0.010000, Accuracy: 91.77%
-- Epoch [5/15], Loss: 0.2692, LR: 0.001000, Accuracy: 92.32%
-- Epoch [6/15], Loss: 0.2552, LR: 0.001000, Accuracy: 92.78%
-- Epoch [7/15], Loss: 0.2531, LR: 0.001000, Accuracy: 92.85%
-- Epoch [8/15], Loss: 0.2513, LR: 0.001000, Accuracy: 92.88%
-- Epoch [9/15], Loss: 0.2496, LR: 0.001000, Accuracy: 92.94%
-- Epoch [10/15], Loss: 0.2479, LR: 0.000100, Accuracy: 92.98%
-- Epoch [11/15], Loss: 0.2467, LR: 0.000100, Accuracy: 93.04%
-- Epoch [12/15], Loss: 0.2465, LR: 0.000100, Accuracy: 93.05%
-- Epoch [13/15], Loss: 0.2463, LR: 0.000100, Accuracy: 93.06%
-- Epoch [14/15], Loss: 0.2461, LR: 0.000100, Accuracy: 93.05%
-- Epoch [15/15], Loss: 0.2459, LR: 0.000010, Accuracy: 93.08%
```




```
-- Epoch [1/15], Loss: 0.2458, LR: 0.000009, Accuracy: 93.08%
-- Epoch [2/15], Loss: 0.2458, LR: 0.000008, Accuracy: 93.08%
-- Epoch [3/15], Loss: 0.2458, LR: 0.000007, Accuracy: 93.08%
-- Epoch [4/15], Loss: 0.2458, LR: 0.000007, Accuracy: 93.08%
-- Epoch [5/15], Loss: 0.2457, LR: 0.000006, Accuracy: 93.08%
-- Epoch [6/15], Loss: 0.2457, LR: 0.000005, Accuracy: 93.08%
-- Epoch [7/15], Loss: 0.2457, LR: 0.000005, Accuracy: 93.08%
-- Epoch [8/15], Loss: 0.2457, LR: 0.000004, Accuracy: 93.09%
-- Epoch [9/15], Loss: 0.2457, LR: 0.000004, Accuracy: 93.08%
-- Epoch [10/15], Loss: 0.2457, LR: 0.000003, Accuracy: 93.09%
-- Epoch [11/15], Loss: 0.2457, LR: 0.000003, Accuracy: 93.08%
-- Epoch [12/15], Loss: 0.2457, LR: 0.000003, Accuracy: 93.08%
-- Epoch [13/15], Loss: 0.2457, LR: 0.000003, Accuracy: 93.08%
-- Epoch [14/15], Loss: 0.2457, LR: 0.000002, Accuracy: 93.08%
-- Epoch [15/15], Loss: 0.2457, LR: 0.000002, Accuracy: 93.09%
```



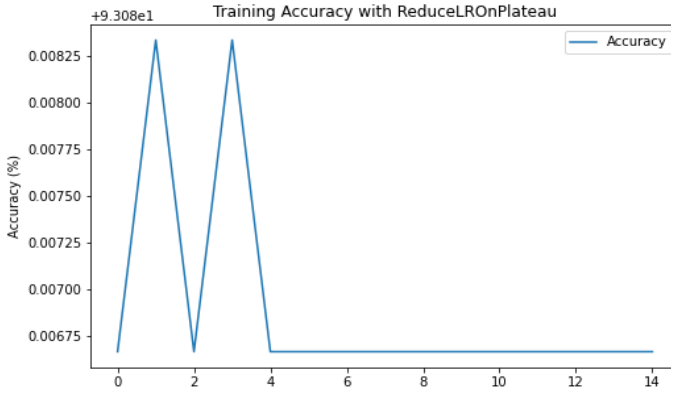
The loss starts at 0.2458 and remains stable throughout the training and ending at 0.2457. A little change has been made due to no longer improvements from the model. The learning rate start extremely low, only at 0.000009 and continues to decrease All the way down to 0.000002 at the final epoch. This type of change leads accuracy to remain stable with only minor increases.

The learning rate model ExponentialLR has only led to very stable accuracy due to the extremely low learning rate. I did assume that it would give me slight differences because I felt that the learning rate was more aimed for tasks where

overfitting is a concern, whereas for MINST dataset it prevented additional learning.

ReduceLROnPlateau reduces the learning rate when the model's performance stops increasing [11]. Here, the model will somehow progress quickly at initially, but subsequently reach a point when accuracy stops improving. When this occurs, ReduceLROnPlateau will reduce the learning rate, allowing the model to make additional advances without being stuck.

```
-- Epoch [1/15], Loss: 0.2457, LR: 0.000002, Accuracy: 93.09%
-- Epoch [2/15], Loss: 0.2457, LR: 0.000002, Accuracy: 93.09%
-- Epoch [3/15], Loss: 0.2457, LR: 0.000002, Accuracy: 93.09%
-- Epoch [4/15], Loss: 0.2457, LR: 0.000000, Accuracy: 93.09%
-- Epoch [5/15], Loss: 0.2457, LR: 0.000000, Accuracy: 93.09%
-- Epoch [6/15], Loss: 0.2457, LR: 0.000000, Accuracy: 93.09%
-- Epoch [7/15], Loss: 0.2457, LR: 0.000000, Accuracy: 93.09%
-- Epoch [8/15], Loss: 0.2457, LR: 0.000000, Accuracy: 93.09%
-- Epoch [9/15], Loss: 0.2457, LR: 0.000000, Accuracy: 93.09%
-- Epoch [10/15], Loss: 0.2457, LR: 0.000000, Accuracy: 93.09%
-- Epoch [11/15], Loss: 0.2457, LR: 0.000000, Accuracy: 93.09%
-- Epoch [12/15], Loss: 0.2457, LR: 0.000000, Accuracy: 93.09%
-- Epoch [13/15], Loss: 0.2457, LR: 0.000000, Accuracy: 93.09%
-- Epoch [14/15], Loss: 0.2457, LR: 0.000000, Accuracy: 93.09%
-- Epoch [15/15], Loss: 0.2457, LR: 0.000000, Accuracy: 93.09%
```



The final result after the training shows the loss being constant at 0.2457 throughout the entire training. The mode is not making further progress. The learning rate, however, is extremely low and only drops from 0.000002 to 0.000000. accuracy stays on 93% on all epochs. This stability indicates that the model hit its maximal accuracy early on and was unable to progress further, potentially because of the relatively low learning rate, which limited its ability to modify. Looks like ReduceLROnPLateau may not be well-suited in this MINST dataset.

Overall, for the MINST dataset, the StepLR was well-suited learning for MINST among the three learning rates. It enables the model to achieve significant progress in the early epochs and then fine-tunes without unnecessary limiting the learning rate. The learning rate model guarantees that learning is effective while still being flexible for further improvement.

With the CIFAR-10 dataset, I used Cyclic Learning Rate (CyclicLR), Cosine Annealing Learning Rate, (CosineAnnealingLR), and OneCycleLR. CyclicLR adjusting the learning rate between a minimum and maximum value to help the model to exit the local minimum or saddle point without skipping the global minimum [12]. CosineAnnealingLR has a strategy that begins with a high learning rate and aggressively reduces it to a number near zero before boosting it again [13]. OneCycleLR increases the learning rate to a peak, then reduces it within a single training cycle.

During the process of training the learning rate models on CIFAR-10 model, I did not get the same amount result as the MINST dataset. I tried to fix things up but could not find the issue. With the cyclicLR learning rate the loss remained constant at 2.3035 on all epochs. the model does not learn effectively. I did expect a loss to decrease as the model remained untouched. The accuracy also remained the same through the process, but the Learning rate did increase from 0.002800 to 0.010000 over the five epochs.

```
Epoch [1/5], Loss: 2.3035, Accuracy: 9.88%, LR: 0.002800
Epoch [2/5], Loss: 2.3035, Accuracy: 9.88%, LR: 0.004600
Epoch [3/5], Loss: 2.3035, Accuracy: 9.88%, LR: 0.006400
Epoch [4/5], Loss: 2.3035, Accuracy: 9.88%, LR: 0.008200
Epoch [5/5], Loss: 2.3035, Accuracy: 9.88%, LR: 0.010000
```

The next learning rate, CosineAnnealingLR gave the same result as CyclicLR. Only the learning rate went from 0.009777 to 0.00500. loss and accuracy remained the same.

```
Epoch [1/5], Loss: 2.3035, Accuracy: 9.88%, LR: 0.009755
Epoch [2/5], Loss: 2.3035, Accuracy: 9.88%, LR: 0.009045
Epoch [3/5], Loss: 2.3035, Accuracy: 9.88%, LR: 0.007939
Epoch [4/5], Loss: 2.3035, Accuracy: 9.88%, LR: 0.006545
Epoch [5/5], Loss: 2.3035, Accuracy: 9.88%, LR: 0.005000
```

Had the same issue with the OneCycleLR. All remained the same from the first epoch to the last one.

I made my research and did changes to my code but could not solve the issue. One of the things I needed to do was to verify the model configuration to ensure the model is set up correctly with necessary complexity to handle the dataset. I saw only increasing numbers on the learning rate but not on the accuracy and loss. I did an examination on the optimizer and data processing to confirm that the optimizer is properly created, but still somehow could not find the issue. I did changes on the epoch, took up to 20 epochs instead, and increased the base_lr to 0.015 and max_lr from 0.01 to 0.02. this is the result:

```
Epoch [1/20], Loss: 2.3035, Accuracy: 9.88%, LR: 0.016000
Epoch [2/20], Loss: 2.3035, Accuracy: 9.88%, LR: 0.017000
Epoch [3/20], Loss: 2.3035, Accuracy: 9.88%, LR: 0.018000
Epoch [4/20], Loss: 2.3035, Accuracy: 9.88%, LR: 0.019000
Epoch [5/20], Loss: 2.3035, Accuracy: 9.88%, LR: 0.020000
Epoch [6/20], Loss: 2.3035, Accuracy: 9.88%, LR: 0.019000
Epoch [7/20], Loss: 2.3035, Accuracy: 9.88%, LR: 0.018000
Epoch [8/20], Loss: 2.3035, Accuracy: 9.88%, LR: 0.017000
Epoch [9/20], Loss: 2.3035, Accuracy: 9.88%, LR: 0.016000
Epoch [10/20], Loss: 2.3036, Accuracy: 9.88%, LR: 0.015000
Epoch [11/20], Loss: 2.3035, Accuracy: 9.88%, LR: 0.016000
Epoch [12/20], Loss: 2.3035, Accuracy: 9.88%, LR: 0.017000
Epoch [13/20], Loss: 2.3035, Accuracy: 9.88%, LR: 0.018000
Epoch [14/20], Loss: 2.3035, Accuracy: 9.88%, LR: 0.019000
Epoch [15/20], Loss: 2.3035, Accuracy: 9.88%, LR: 0.020000
Epoch [16/20], Loss: 2.3035, Accuracy: 9.88%, LR: 0.019000
Epoch [17/20], Loss: 2.3035, Accuracy: 9.88%, LR: 0.018000
Epoch [18/20], Loss: 2.3035, Accuracy: 9.88%, LR: 0.017000
Epoch [19/20], Loss: 2.3035, Accuracy: 9.88%, LR: 0.016000
Epoch [20/20], Loss: 2.3035, Accuracy: 9.88%, LR: 0.015000
```

Still, the model is not improving since the loss remains 2.3035 throughout all 20 epochs. The accuracy also remains the same, but the learning rate fluctuates within the set range, rising and decreasing as determined by cyclicLR learning rate. it alternates between 0.015000 and 0.019000 throughout the course of 20 epochs.

Acknowledgment

This experiment emphasizes the necessity of choosing appropriate optimization, learning rate schedulers, and augmentation techniques for MINST dataset using NLP as the model and CIFAR-1 with CNN as the model. The experiment indicates that current optimizers, adaptive learning rate schedulers, and augmentation techniques will improve and impair performance and efficiency.

References

[1] Krizhevsky, A., «Learning multiple layers of features from tiny images", 2009. [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html).

[2] Cloudfactory, "StepLR Scheduler Documentation," 2024. [Online]. Available: https://wiki.cloudfactory.com/docs/mp-wiki/scheduler/steplr)

[3] Cloudfactory, "ExponentialLR Scheduler Documentation," 2024.[Online]. Available: https://wiki.cloudfactory.com/docs/mp-wiki/scheduler/exponentiallr)

[4] GeeksforGeeks, "Stochastic Gradient Descent (SGD)", 2024.[Online]. Available: https://www.geeksforgeeks.org/ml-stochastic-gradient-descent-sgd/

[5] Roy, B., "Adam vs. SGD: Let's drive in" 2019. [Online]. Available: https://medium.com/@Biboswan98/optim-adam-vs-optim-sgd-lets-dive-in-8dbf1890fbdc

[6] GeeksforGeeks, "RMSprop Optimizer in Deep Learning", 2024.[Online]. Available: https://www.geeksforgeeks.org/rmsprop-optimizer-in-deep-learning/).

[7] Agarwal, P., "Adam Optimization Explained," 2023. [Online]. Available: https://builtin.com/machine-learning/adam-optimization

[8] Keras, "AdamW Optimizer Documentation", [Online]. Available: https://keras.io/api/optimizers/adamw/

[9] Keras, "Adamax Optimizer Documentation", [Online]. Available: https://keras.io/api/optimizers/adamax/)

[10] Tripathi, S., "Adagrad Optimizer Explained," 2024. [Online] available: https://www.datacamp.com/tutorial/adagrad-optimizer-explained

[11] Cloudfactory, "ReduceLROnPlateau Scheduler Documentation", 2024. [Online]. Available: https://wiki.cloudfactory.com/docs/mp-wiki/scheduler/reducelronplateau

[12] Cloudfactory, "CyclicLR scheduler Documentation," 2024. [Online]. Available: https://wiki.cloudfactory.com/docs/mp-wiki/scheduler/cycliclr

[13] Cloudfactory, "Paraphrasing Tool Documentation", 2024. [Online]. Available: https://quillbot.com/paraphrasing-tool