Creation of FinalNet CNN for CIFAR10
CSE532 Final Report

Alexander Habegger

Task Definition

Topic A

Design a new CNN model for the task in HW7. Your model should be different from YourNet and LeNet. Your proposed model's maximum number of layers should be 10. You must implement your proposed model and achieve better classification accuracy than in the HW7. Next, explain what you modified to improve the performance (you can start changing YourNet based on what you learned in Chapters 5 and 6). Finally, evaluate the performance.

Approaches Taken

Non-Contribution Changes in FinalNet

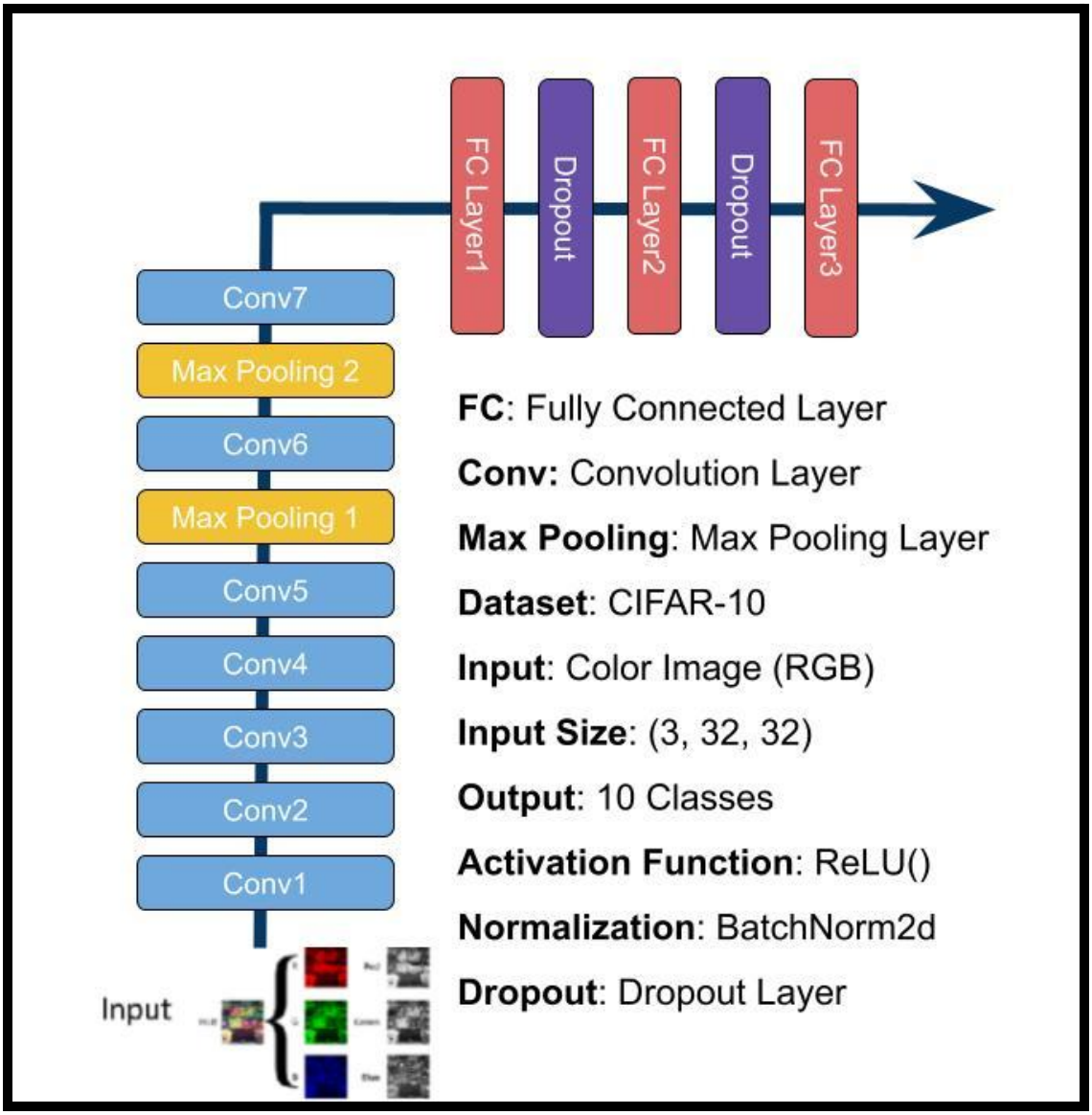
- Modify the starting learning rate
- Modify the batch size

Contribution Changes in FinalNet

- Added Dropout Layers
- Added Batch Normalization Layers
- Changed Optimizer to Adam
- Different CNN Architecture
- Added Learning Rate Scheduler

FinalNet Architecture

Structure of Layers



FinalNet Architecture

Layer Description

Layer	Kernel Size	Stride	Padding	Number of Filters
Conv1	11	1	5	16
Conv2	7	1	3	32
Conv3	5	1	2	32
Conv4	5	1	2	64
Conv5	5	1	3	128
Max Pool1	2	2	0	-
Conv6	5	1	0	256
Max Pool2	2	2	0	-
Conv7	15	1	0	320

Parameter & Output Description

Layer	# of Parameters	Output Size
Conv1	(11 * 11 * 3 + 1) x 16 = 5,840	16 x 32 x 32
Conv2	(7 * 7 * 16 + 1) x 32 = 25,120	32 x 32 x 32
Conv3	(5 * 5 * 32 + 1) x 32 = 25,632	32 x 32 x 32
Conv4	(5 * 5 * 32 + 1) x 64 = 51,264	64 x 32 x 32
Conv5	(5 * 5 * 64 + 1) x 128 = 204,928	128 x 34 x 34
Max Pool1		128 x 17 x 17
Conv6	(5 * 5 * 128 + 1) x 256 = 819,456	256 x 13 x 13
Max Pool2		256 x 6 x 6
Conv7	(15 * 15 * 256 + 1) x 320 = 1,843,200	320 x 1 x 1

Implementation of Model

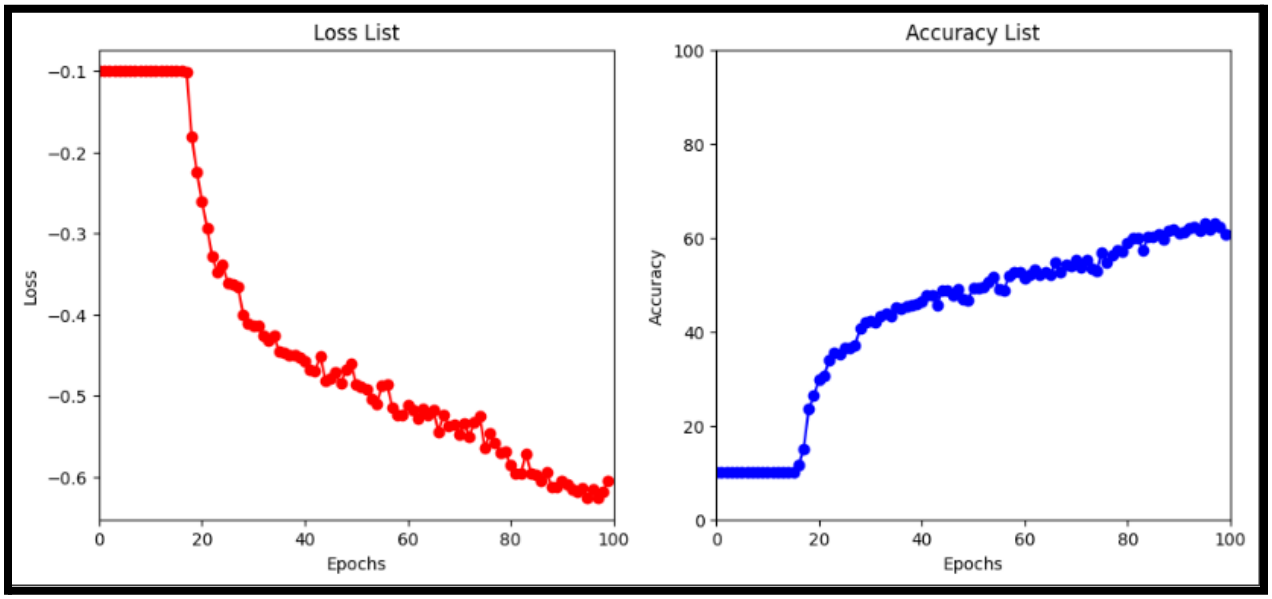
Step 1 Grab Dataset

```python
# Import Dependencies
from __future__ import print_function
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
import time

# Preparing for Data
print('==> Preparing data..')

#  Training Data Augmentation
transform_train = transforms.Compose([
    transforms.RandomCrop(32, padding=4),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)),
])

# Testing Data Preparation
transform_test = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)),
])

# Create Training and Testing Sets
trainset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform_train)
train_loader = torch.utils.data.DataLoader(trainset, batch_size=128, shuffle=True)
testset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=transform_test)
test_loader = torch.utils.data.DataLoader(testset, batch_size=100, shuffle=False)
```

Using the PyTorch framework, the program creates training and testing set for the model on the CIFAR-10 dataset. First, the program imports the necessary dependencies and augments the training set to prevent overfitting. Then, the program applies the transformations and normalizations to make them easier to work with. Finally, the program loads them into data loaders to use later for the model.

Implementation of Model

Step 2 Start with YourNet (HW7)



The final results of YourNet with the hyperparameters of 100 epochs, 32 batch size, the learning rate of 0.0021206421108366574, and an SGD optimizer with a weight_decay of 5e-4 and a momentum of 0.85 is an accuracy of 61%. The YourNet model with these hyperparameters is the best model from HW7. This project aims to create a new modified model with more accuracy. So lots of changes need to be made to the model. Let's start with modifying the architecture.

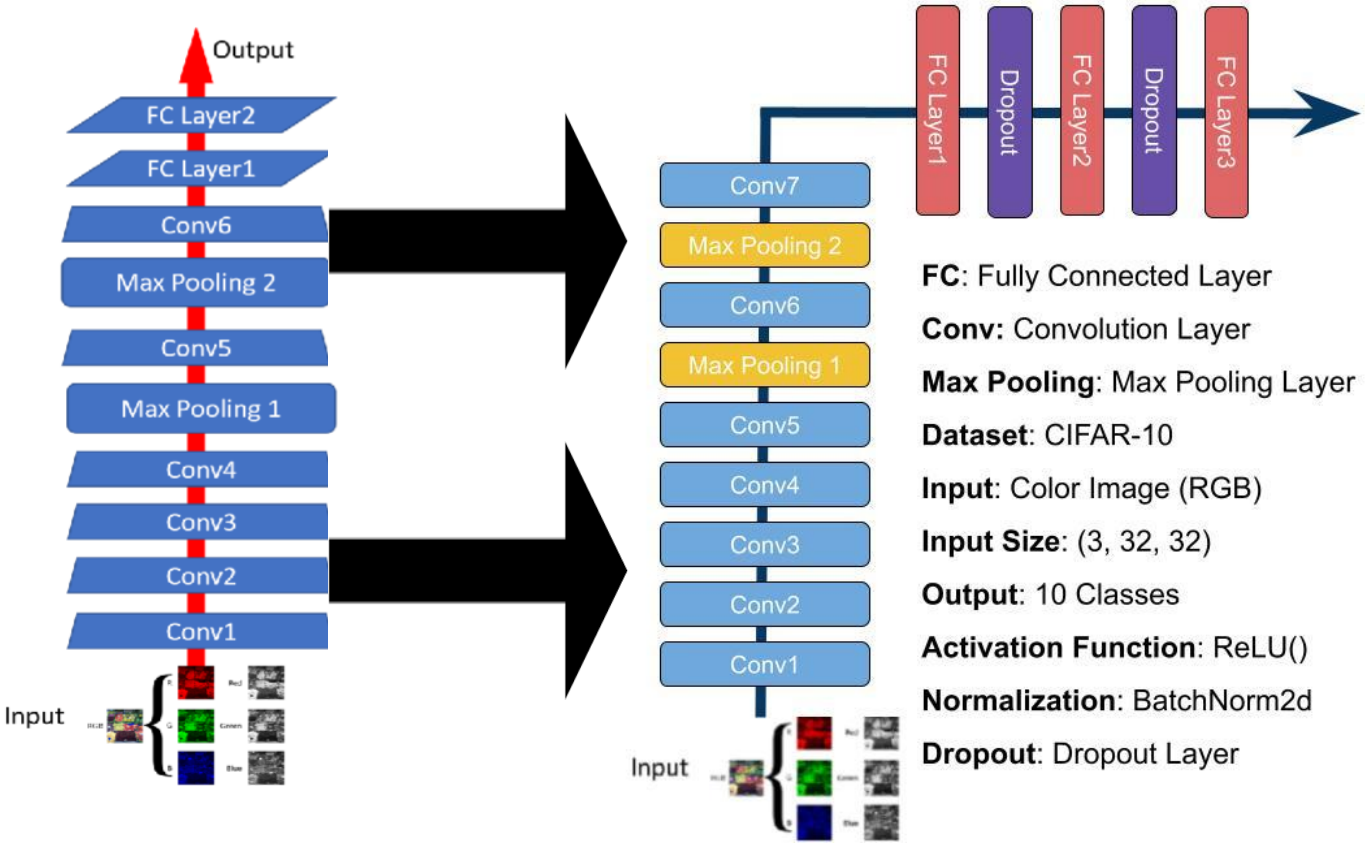
Implementation of Model

Step 3 Modify Architecture

Here are the differences between the FinalNet and YourNet architectures:

- FinalNet contains batch normalization layers after each convolution layer except the last one, while YourNet has no batch normalization layers.
- FinalNet has 7 convolution layers, while YourNet has 6.
- FinalNet has different numbers of output channels for different convolution layers 8, 16, 16, 32, 64, 128, and 160, while YourNet has 8, 16, 16, 16, 16, and 160 output channels for its convolution layers.
- FinalNet's fourth convolution layer has a padding of 2, while YourNet has 0.
- FinalNet has three fully connected layers with dropout layers, while YourNet has two fully connected layers without any dropout layers.

These changes were created to increase the ability of the model to pick up on local patterns and not overfit. More convolutional layers have been seen to increase predictive ability (Jmour et al. 2018). The addition of batch normalization layers (Santurkar et al. 2018). The increasing parameters also can help in accuracy, while the dropout layers between the fully connected layers help prevent overfitting (Srivastava et al. 2014). These changes will likely increase the predictive ability of a model.



Implementation of Model

Step 4 Add Learning Rate Scheduler and New Optimizer

Add a Learning Rate Scheduler

```
# Gamma is added as a hyperparameter
gamma = 0.1 # Default Value

# Define the Learning Rate Scheduler - MultiStepLR
scheduler = optim.lr_scheduler.MultiStepLR(optimizer, milestones=[15, 30], gamma=gamma)

for epoch in range(1, epochs + 1):
    train(model, device, train_loader, optimizer, epoch)
    test_loss, correct = test(model, device, test_loader)
    scheduler.step() # This line implements the schedule after every epoch
```

A learning rate scheduler is added to a neural network model to adjust the learning rate. This allows for adaptive training, improving convergence speed and overall model accuracy. In addition, it helps prevent escaping from local minima or plateaus.

Choosing a New Optimizer

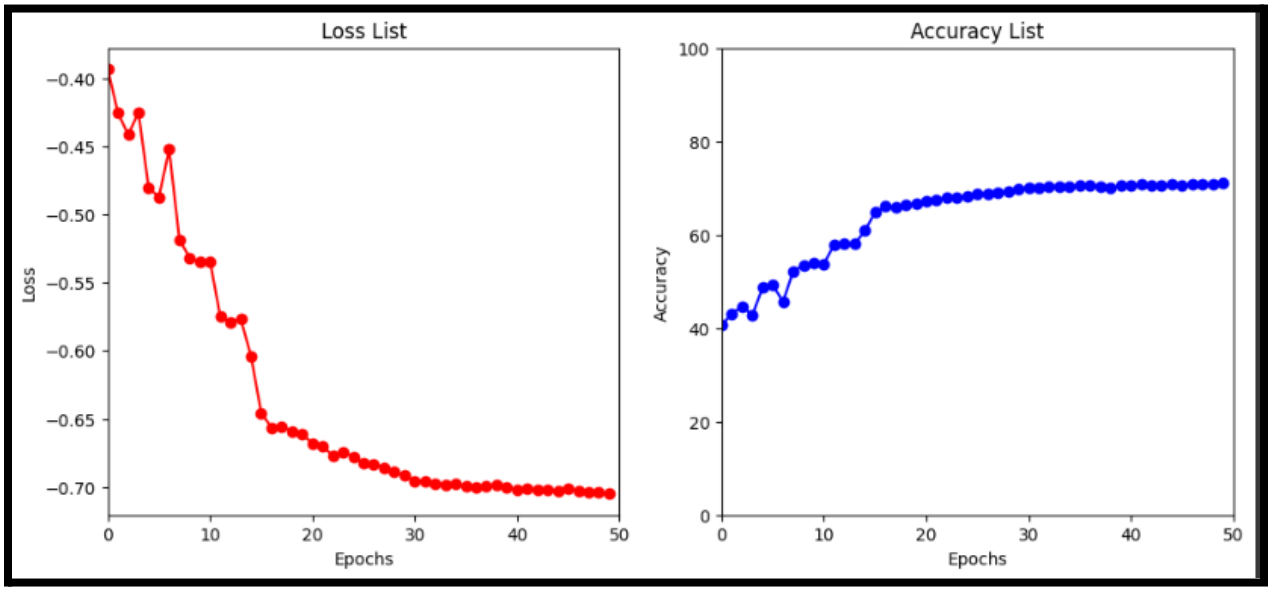
```
# Changing the Optimizer from SGD to Adam
optimizer = optim.Adam(model.parameters(), lr=lr, weight_decay=5e-4)
```

The decision to replace the Stochastic Gradient Descent (SGD) optimizer with the Adaptive Moment Estimation (Adam) optimizer is practical. The reasoning is that Adam uses momentum and adaptive learning rates to induce faster convergence and improve model accuracy. Also, Adam is less sensitive to hyperparameter choices allowing for less optimization of hyperparameters to create a successful model.

Implementation of Model

Step 5 Optimize Hyperparameters

```
import optuna
def objective(trial):
    # Define the hyperparameters
    lr = trial.suggest_loguniform("lr", 1e-5, 1e-3)
    batch_size = trial.suggest_categorical("batch_size", [128, 256, 518])
    gamma = trial.suggest_categorical("gamma", [0.08, 0.1, 0.12])
    droprate = trial.suggest_categorical("droprate", [0.4, 0.5, 0.6])
```



This step aims to optimize the hyperparameters of batch size, learning rate, gamma, and droprate. To achieve this goal, Optuna will be used. Optuna is for hyperparameter optimization of the neural network (Akiba et al. 2019). In the program, the objective function trains the model with different hyperparameters suggested by Optuna for multiple trials. Sadly the program crashed at Trial 23, but all trials were recorded and analyzed. The best trial before it crashed was Trial 22 had the best accuracy. The trial had a learning rate of 0.0021206421108366574 and a batch size of 32.

```
[I 2023-04-15 05:42:34,653] Trial 22 finished with value: -0.6952951594352722
```

Implementation of Model

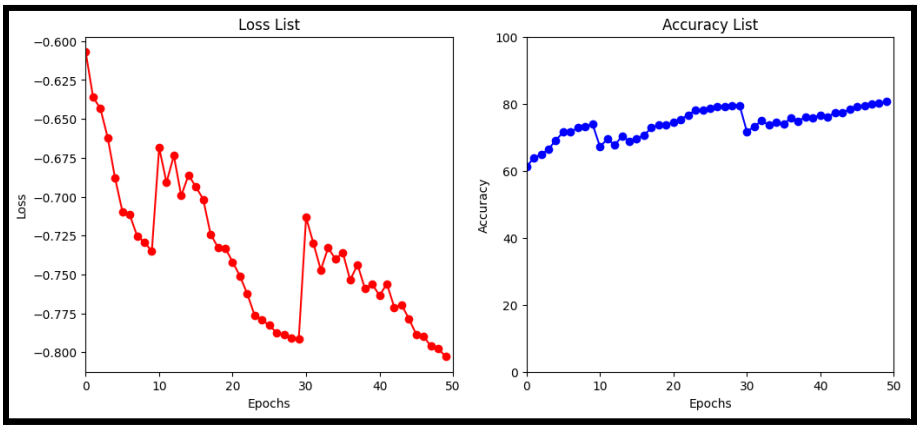
Step 6 Change the Learning Rate Scheduler

```
scheduler = optim.lr_scheduler.CosineAnnealingWarmRestarts(optimizer, T_0=10, T_mult=2, eta_min=1e-6)
```

As seen in Step 5, the model accuracy plateaus after epoch 20. To counter this, the search for different learning rate schedulers began. The result was CosineAnnealingWarmRestarts as the learning rate scheduler (Leonie 2020). It provides a smoother and more cyclic learning rate schedule. The cyclical nature allows for the ability to escape local minima that can impede future learning. Furthermore, the warm restarts allow faster solution space exploration, which may improve model performance. The result is the finalized version of FinalNet.

Summary

Performance of FinalNet Best Model



Here are the end accuracy of the developed models:

- (YN-7) YourNet HW7 Best Model with Optimized Hyperparameters = 61%
- (FN-22) FinalNet Trial 22 with Optimized Hyperparameters = 71%
- (FN-Alpha) FinalNet Best Model with New Learning Rate Scheduler = 81%

Overall the performance of the model shows a clear improvement on previous models. The FN-Alpha model's cyclical learning rate will make large breakthroughs and slow progress. Also, the warm resets on the learning rate schedule cause the increase in loss and decrease in accuracy seen in epochs 10 and 30. I attribute most of the gains in performance due to the drop layers preventing the overfitting seen in YN-7, and the CosineAnnealingWarmRestarts learning rate scheduler preventing stagnant loss and accuracy due to plateau and local minima seen in FN-22. The initial accuracy after epoch 1 was a surprising 61% this could be attributed to the more extensive architecture and number of parameters seen in FN-22 and FN-Alpha compared to YN-7.

Analysis & Discussion

The developed model is a CNN for image classification on the CIFAR-10 dataset called FinalNet. CIFAR-10 is a dataset that comprises 60,000 32 by 32 color images across ten different classes. The FinalNet architecture consists of a feature extractor and a classifier. The feature extractor contains a series of convolutional, batch normalization, and activation layers along with max-pooling layers. The classifier contains three fully connected layers with ReLU activations and dropout layers.

The code defines functions for the training and testing phases to train the model and initializes hyperparameters of a batch size of 128, epochs of 50, a learning rate of 0.0021206421108366574, and weight decay of 5e-4. The model is trained using the Adam optimizer and a learning rate scheduler of CosineAnnealingWarmRestarts. Finally, the model is trained and tested for each epoch, with the results recorded in lists for loss and accuracy. The model is saved at the end of the training process, and the loss and accuracy values for each epoch are plotted using matplotlib. An analysis of each change in the model is discussed in the Implementation of Model Section, where each step is justified.

Resources

Citations

N. Jmour, S. Zayen and A. Abdelkrim, "Convolutional neural networks for image classification," 2018 International Conference on Advanced Systems and Electric Technologies (IC_ASET), Hammamet, Tunisia, 2018, pp. 397-402, doi: 10.1109/ASET.2018.8379889.

Santurkar, S., Tsipras, D., Ilyas, A., & Madry, A. (2018). How does batch normalization help optimization?. *Advances in neural information processing systems*, *31*.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, *15*(1), 1929-1958.

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta,and Masanori Koyama. 2019. Optuna: A Next-generation Hyperparameter Optimization Framework. *In KDD*.

Monigatti, Leonie. “A Visual Guide to Learning Rate Schedulers in Pytorch.” *Medium*, Towards Data Science, 6 Dec. 2022, https://towardsdatascience.com/a-visual-guide-to-learning-rate-schedulers-in-pytorch-24bbb262c863.

Code

https://colab.research.google.com/drive/1ZBVs5SRDbNJZVrwdpBqLE3yF6geZPOnU?usp=sharing

Software Used

Google Colab for Code and Testing

Google Scholar for Citations

EasyBib for Website Citations

Google Drive for Storage

Optuna for Hyperparameter Optimization

StackOverflow for Errors

Google Drawings for Diagram

Grammarly for Proofreading

Dependencies Listed in Code