Exploring Residual Neural Net Proficiency on CIFAR-10

Satyam Chatrola (sc10247), Victor Pou (vmp2063), Xinhai Leng (xl3009)
GitHub Link: https://github.com/Nightshade14/Deep-Learning-Midterm-project

Abstract

This project explores various neural network architectures to determine their efficacy in classifying images of the CIFAR-10 dataset. We experimented with several configurations of ResNet and modifications in activation functions like ReLU, ELU, and SiLU (Swish). Our methodology involved systematic training and evaluation of each model under controlled conditions, focusing on understanding the impact of architectural differences and learning rate adjustments. Results indicated that specific adjustments in the learning rate scheduler significantly affected model accuracy. The best-performing model achieved an accuracy of 92% on the validation set and 82% on the Kaggle no-label test dataset, highlighting the importance of carefully chosen hyper-parameters in neural network performance. This study underscores the potential of advanced neural architectures in image recognition tasks and provides insights into the optimization of deep learning models for improved accuracy. The repository of the project is hosted on Github (Satyam Chatrola sc10247)

Introduction

As the field of computer vision has made significant advances in the past decade, the field has found endless applications in various fields, such as in healthcare, augmented reality, face recognition, and autonomous vehicles. As a result of its extreme relevancy, each step in this field of inference has the potential to be world changing on all fronts of society. The introduction of Residual Neural Networks (ResNets) had a particularly substantial contribution to computer vision as it solved the vanishing gradient issue that had been hindering the development of effective deep learning models. (He et al. 2015)

This study explores the practices of making an effective, light ResNet. 'Light', as in the model being used has been limited to 5 million parameters. The model is tested on the CIFAR-10 dataset. The dataset is a compilation of images of vehicles and animals. It's often used to test the object classification capabilities of the model. By leveraging dropout, a scheduler, and data transformations, the model gave satisfactory results on the classification of the CIFAR-10 dataset.

Related Work

ResNet

Before the introduction of ResNets, the vanishing gradient problem would increasingly degrade the effectiveness of deep learning networks the deeper they became. The study, Deep Residual Learning for Image Recognition, conducted by He et al. proposed the use of skip connections, as well as an architecture composed of residual building blocks. The building blocks perform convolutions and their outputs are sent as an input to the next block and a block a couple layers ahead, performing the skip connection. The skip connection allows gradients to backpropagate through the network more efficiently by flowing through the skips and eluding the issue caused by propagating through each layer and diminishing gradients to infinitesimally small values. This study utilizes the ResNet model as a strong foundation to modify and improve in regards to its image classification performance on the CIFAR-10 dataset.(He et al. 2015)

CIFAR-10

CIFAR-10 is a dataset of 60,000 images of airplanes, automobiles, birds, cats, dogs, frogs, deer, horses, ships, and trucks constructed by Alex Krizhevsky for image classification testing. The dataset is used to test the performance of the proposed model. (Krizhevsky, Nair, and Hinton)

Pytorch

PyTorch is an open-source machine learning library developed by Facebook's AI Research lab, primarily used for Deep Learning applications in vision, text and audio domain. It was employed to construct the ResNet in this study. It also provides several utilities that improved the performance of the proposed model: dropout, transforms, StepLR, and SGD. (Paszke et al. 2019)

Dropout is a method of regularization, used to decrease the model's ability to over fit to the data by excluding a certain portion of weights from being adjusted by the optimizer for each epoch. Dropout keeps the model from over adjusting weights it might otherwise want to place great emphasis on. The base ResNet constructed for this study tends to overfit without regularization methods like dropout to keep it in check. (Srivastava et al. 2014; Paszke et al. 2019)

Pytorch's transforms provides the tools to augment and diversify the images being fed into training through a stochastic combination of rotating, flipping, cropping, scaling, and/or color adjustments. The process forces the model to focus on invariant features that truly define the class space of the sample. The proposed model uses data transformations to boost performance.

StepLR is a scheduler, used to decay the learning rate as learning advances. Reducing the rate throughout learning allows for a quick yet optimal convergence when optimizing the model and its objective function. StepLR is used in the proposed model to aid with optimization. (nep 2023; Paszke et al. 2019)

SGD is a function in Pytorch used to perform stochastic gradient descent (SGD), a minimization objective function. SGD, through a stochastic process, gathers a single data point or small group of of data points to perform gradient descent on and optimize a model's performance. ADAM is often used as an alternative objective function, however, to optimize the proposed model SGD was used for its advantageous speed and its performance on the CIFAR-10 dataset. (Bottou, Curtis, and Nocedal 2018; Paszke et al. 2019)

Methodology

This section details the systematic approach to developing a residual neural network for image classification on the CIFAR-10 dataset, emphasizing robustness and accuracy through architectural choices and data handling strategies.

Data Preprocessing and Augmentation

We implemented several data augmentation techniques on the training dataset to enhance the model's generalization ability. These transformations, managed through the `torchvision.transforms` module, included random horizontal flips, random cropping, and normalization. Notably, these augmentations were not applied to the validation dataset, which was only normalized to ensure consistency in evaluation.

Model Architecture

Our model architecture leverages the ResNet framework, utilizing residual blocks that help mitigate the vanishing gradient problem, thus enabling the training of deeper networks. The model was created with the help of PyTorch. The Pytorch code for the architecture is inspired from the GitHub repository of Kuang Liu (kuang liu 2021). The repository provides flexible, customizable and robust code for creating a ResNet with Pytorch. There are 4 layers in the network with 3 Residual blocks in each. Each residual block comprises sequential convolutional layers, followed by batch normalization and ReLU activation functions. To combat overfitting, dropout layers with a rate of 0.6—determined as optimal through iterative testing—were integrated following the activation functions, as suggested by Srivastava et al. (Srivastava et al. 2014). The model architecture is as in the Figure 1.

Training

The network was trained using Stochastic Gradient Descent (SGD) with momentum, across multiple epochs. A learning rate scheduler was employed to adjust the rate dynamically, significantly enhancing convergence characteristics by allowing the model to settle at more optimal local minima as training progressed, a method supported by literature on learning rate adjustments (nep 2023). The dropout strategy was particularly effective in addressing initial overfitting, underscoring the importance of dropout in training deep neural networks.

Evaluation

Performance evaluation was primarily focused on the accuracy metric, assessing the model's ability to classify images within the validation set accurately. This set served as a stand-in for test conditions to estimate model performance in a real-world scenario.

The systematic exploration of different configurations and training regimens was instrumental in identifying an optimal setup that improved prediction accuracy and effectively addressed overfitting, underscoring the critical role of hyperparameter tuning and architectural considerations in deep learning models.

Results

Performance Optimization

Our project aimed to optimize a neural network for classifying images from the CIFAR-10 dataset. Through a series of methodical enhancements to model architecture and hyperparameter adjustments, we achieved substantial improvements in accuracy.

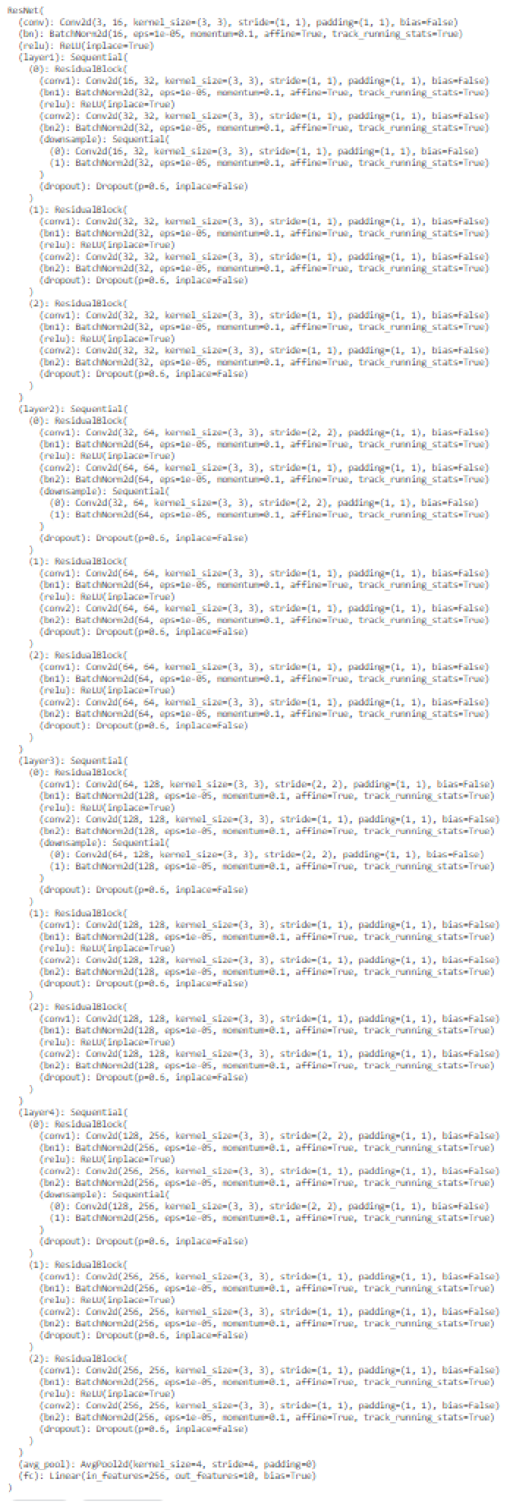
Initially, the model's accuracy was about 8%. With targeted modifications, including architectural changes and hyperparameter optimization, the accuracy increased to around 40%. Further refinements led to accuracy levels reaching between 70% and 80%. Key adjustments that significantly impacted performance included optimizing the dropout rate to 0.6 and setting batch sizes to 128 for training and 100 for testing. These parameters helped mitigate overfitting and enhanced computational efficiency.

Learning Rate Scheduling

The learning rate scheduling was crucial to the model's training dynamics. We implemented a Stochastic Gradient Descent (SGD) optimizer with an initial learning rate of 0.1, reduced by a factor of 0.1 every 20 epochs. This approach allowed for finer adjustments in the model's weight updates in later training phases, avoiding potential overshooting of minima. The Loss per epoch graph is shown in Figure 2.

Final Outcome and Performance Analysis

The final model configuration achieved a peak accuracy of 92%, confirming our approach's effectiveness. The choice of SGD as the optimizer was particularly beneficial, as it



```
ResNet(
  (conv): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (bn): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (layer1): Sequential(
    (0): ResidualBlock(
      (conv1): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
      (dropout): Dropout(p=0.6, inplace=False)
    )
    (1): ResidualBlock(
      (conv1): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (dropout): Dropout(p=0.6, inplace=False)
    )
    (2): ResidualBlock(
      (conv1): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (dropout): Dropout(p=0.6, inplace=False)
    )
  )
  (layer2): Sequential(
    (0): ResidualBlock(
      (conv1): Conv2d(32, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(32, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
        (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
      (dropout): Dropout(p=0.6, inplace=False)
    )
    (1): ResidualBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (dropout): Dropout(p=0.6, inplace=False)
    )
    (2): ResidualBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (dropout): Dropout(p=0.6, inplace=False)
    )
  )
  (layer3): Sequential(
    (0): ResidualBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
      (dropout): Dropout(p=0.6, inplace=False)
    )
    (1): ResidualBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (dropout): Dropout(p=0.6, inplace=False)
    )
    (2): ResidualBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (dropout): Dropout(p=0.6, inplace=False)
    )
  )
  (layer4): Sequential(
    (0): ResidualBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
      (dropout): Dropout(p=0.6, inplace=False)
    )
    (1): ResidualBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (dropout): Dropout(p=0.6, inplace=False)
    )
    (2): ResidualBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (dropout): Dropout(p=0.6, inplace=False)
    )
  )
  (avg_pool): AvgPool2d(kernel_size=4, stride=4, padding=0)
  (fc): Linear(in_features=256, out_features=10, bias=True)
)
```

Figure 1: Model Architecture [4.7M parameters]

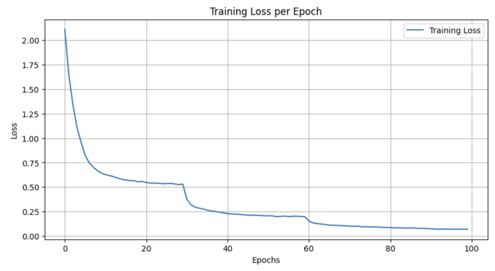


Figure 2: Training loss per epoch

is known for its robustness and efficacy in managing noise, which is crucial for training deep neural networks on large datasets like CIFAR-10 (Paszke et al. 2019).

Graphical representations of the accuracy progression and the impact of different dropout rates visually substantiate the improvements and the optimization process. A detailed error analysis indicated that most classification errors occurred in categories with visually similar objects, suggesting areas for future model refinement.

This study highlights the capabilities of advanced neural architectures in image classification tasks and emphasizes the critical role of precise hyperparameter tuning in achieving optimal model performance. Future work could explore alternative architectures, learning rate schedules, and applications to different datasets.

Future Directions

Future investigations could include testing the model with other optimization algorithms to compare performance outcomes or applying the trained model to similar image recognition tasks outside the CIFAR-10 dataset. Further studies might also explore the effects of even lower dropout rates or different activation functions on model performance.

Discussion

Interpretation of Results

The systematic adjustments to our neural network, particularly within the ResNet architecture, significantly enhanced its ability to classify images from the CIFAR-10 dataset. Introducing a dropout rate of 0.6 proved critical in stabilizing training and mitigating overfitting, evident from the loss trends observed during the training phases.

Choice of Model Architecture

ResNets have many possibilities for optimization and customization. The current architecture strikes a balance between Narrow and Wide ResNets, overcoming each other's flaws. Wide networks have relatively more parameters and are prone to overfitting but can model complex patterns whereas Narrow models provide better generalization but are prone to underfitting. During out experimentation phase, we discovered a model that had 99% train accuracy, 93% validation accuracy but only 45% accuracy on test set (Kaggle). This was an overfitting problem. Making the network deeper mitigated the issue.

The other notable point is that the images in the dataset are 32x32 in size whereas ResNets are generally designed for Imagenet whose images are relatively significantly larger, having more information per image. So, the size of the kernels need to be decreased from 7x7 to 3x3. We also adopted down-sampling the image in order to match the dimensions of the image and the input constraints of the network layers.

Impact of Dropout

Before integrating dropout, our model exhibited increasing loss at around 30 epochs, suggesting the onset of overfitting at this stage. However, with the inclusion of dropout, this threshold extended to approximately 60 epochs. This shift indicates that dropout effectively delayed overfitting, allowing the model to learn from the training data for a more extended period without significant performance degradation. This observation underscores the utility of dropout in enhancing the generalization capabilities of deep learning models, particularly in complex image classification tasks.

Theoretical Implications

Applying a learning rate scheduler in our project underscores its theoretical importance in optimizing deep learning models. By employing a step-based reduction strategy—where the learning rate is reduced by a factor of 0.1 every 30 epochs—we could fine-tune the convergence behavior of our neural network. This methodologically controlled decay of the learning rate is significant for enhancing model performance by allowing finer adjustments in the later stages of training and for its implications in understanding learning dynamics.

Theoretically, the learning rate scheduler simulates an annealing process where the 'temperature' of the model's parameter adjustments cools down, preventing disruptive changes to weights that have already begun to stabilize. This approach is grounded in the principles of simulated annealing, a technique borrowed from statistical mechanics, which is often applied to optimization problems to escape local minima and approach a global minimum more effectively.

Our results validate the hypothesis that slower adjustments in the learning rate can lead to better generalization and performance, especially as the model begins to converge. This has implications for training deeper networks on more complex datasets where the risk of overshooting optimal weights becomes more pronounced. Future studies could explore the quantitative impact of different scheduling algorithms on training speed and stability, potentially leading to more adaptive and dynamic scheduling techniques that respond in real-time to the model's state during training.

Limitations and Challenges

Overfitting remains a challenge despite the extensive efforts to refine our model, including adjustments in dropout rates, modifications of layer configurations, and optimization of batch sizes. This persistent issue highlights the complex nature of model training, where not all overfitting can be alleviated with standard regularization techniques. Each strategy, from adjusting dropout to redesigning layers, brings nuances to the training dynamics, yet none offered a complete solution to overfitting within our experiments. The limitations of our approach are notably pronounced when considering the potential variance in results across different datasets or under varied training conditions. Our strategies, while effective to a degree, might not universally prevent overfitting in other contexts or with datasets having different characteristics from CIFAR-10. This suggests a need for more adaptive or novel regularization methods to respond dynamically to the model's learning state and the specific data it encounters.

Conclusion

This project demonstrated the effectiveness of advanced neural network architectures, specifically ResNet, in addressing the challenge of image classification on the CIFAR-10 dataset. Through meticulous experimentation with different configurations and optimizations, we significantly improved model performance, culminating in a final accuracy of 92%.

Our study confirmed the critical importance of hyperparameter tuning, particularly in applying dropout and learning rate scheduling, to combat overfitting and enhance the model's ability to generalize. While dropout helped delay overfitting, allowing the model to learn for an extended number of epochs before experiencing performance degradation, the learning rate scheduler facilitated finer control over the training process, leading to more stable convergence.

Despite these advancements, challenges such as residual overfitting highlight the need to explore model architecture and training strategies continuously. Future work could explore alternative regularization techniques, more dynamic adaptive learning rate algorithms, or novel architectural innovations to enhance further the robustness and accuracy of deep learning models in image classification tasks.

In conclusion, our findings contribute to the broader understanding of deep learning optimization techniques and their practical implications, providing a foundation for future research.

References

2023. How to Choose a Learning Rate Scheduler. Accessed: 10-April-2024.

Bottou, L.; Curtis, F. E.; and Nocedal, J. 2018. Optimization Methods for Large-Scale Machine Learning. arXiv:1606.04838.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2015. Deep Residual Learning for Image Recognition.

Krizhevsky, A.; Nair, V.; and Hinton, G. ???? CIFAR-10 (Canadian Institute for Advanced Research).

kuang liu. 2021. PyTrorch-CIFAR. https://github.com/kuangliu/pytorch-cifar/.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; Desmaison, A.; Kopf, A.; Yang, E.; DeVito, Z.; Raison, M.; Tejani, A.; Chilamkurthy, S.; Steiner, B.; Fang, L.; Bai, J.;

and Chintala, S. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems*, 8024–8035. Curran Associates, Inc.

Satyam Chatrola (sc10247), X. L. x., Victor Pou (vmp2063). 2024. Deep Learning Mid-term. https://github.com/Nightshade14/Deep-Learning-Midterm-project/.

Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15(1): 1929–1958.

Code repository

https://github.com/Nightshade14/Deep-Learning-Midterm-project.git