Evaluation of CNN Models on CIFAR-10: Study of ResNet-18 and VGG-16

Jiafeng Wu
COGS Department
A16916745

Zihao Yang
ECE Department
A16751774

Abstract

This project focuses on the performance of two pretrained convolutional neural network (CNN) models—ResNet-18 and VGG-16—on the CIFAR-10 image classification task. Our goal was to see how model types (ResNet-18 vs. VGG-16), activation function (ReLU vs. LeakyReLU), learning rate (0.001 vs. 0.0001), and number of training epochs (10 vs. 20) affect the performance of convolutional neural networks on the CIFAR-10 classification accuracy. We tested different settings and tracked metrics like accuracy, precision, recall, and F1-score to find out what provides better results. ResNet-18 with LeakyReLU, 20 epochs, Learning Rate of 0.0001 gave the best results overall, while VGG-16 performed slightly better with LeakyReLU. Deeper models like ResNet-18 learned more useful features but needed more training.

1 Introduction

CIFAR-10 is a common image classification dataset that has 60,000 pictures categorized into ten categories, including airplanes, vehicles, and animals. Its size and diversity make it useful for understanding how well deep learning models can generalize.

ResNet and VGG are two widely used convolutional neural network architectures with individual advantages and functions. VGG is commonly used when learning the basics of CNN and doing preliminary experiments because of its straightforward and consistent structure, which uses stacks of 3x3 convolutional layers followed by max pooling. Its simple architecture requires it includes a lot of parameters, making it resource intensive and slower to train and infer, especially on more complicated tasks. ResNet allows networks to go deeper. It has faster convergence in training which improves performance on complex tasks. ResNet tends to generalize better with less fine-tuning than VGG.

In this project, we look at how activation functions (ReLU vs. LeakyReLU), learning rate (0.001 vs. 0.0001), and training duration (10 vs. 20 epochs) influence model performance. We measure

outcomes using metrics like test accuracy, precision, recall, and F1-score to better understand which combinations perform better under our fixed setup using SGD as the optimizer.

2 Methodology

2.1 Dataset

In our project, we relied on the CIFAR-10 dataset for the image classification task. CIFAR-10 consists of 60,000 32x32 color images in 10 categories: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. We used 50,000 images for training and validation (split 80/20), and 10,000 for testing. During each training run, the model processed data in mini-batches of 32 and updated weights after each batch. A progress bar was displayed for each epoch.

We did data augmentation to improve generalization and reduce overfitting. These transformations were implemented using PyTorch's transforms module:

- **RandomHorizontalFlip**: This function creates shifts in object orientation by randomly flipping pictures horizontally.
- **RandomCrop with Padding**: Introduces diversity in object location by randomly cropping the image back to 32×32 pixels after adding a 4-pixel border around it.
- Images are converted to PyTorch tensors for model processing using ToTensor.
- **Normalization**: To match with the pretrained model, ImageNet's mean and standard deviation values—0.485, 0.456, 0.406 for the mean and 0.229, 0.224, 0.225 for the standard deviation—are used.

A batch size of 32 was used during training, meaning that the model processed 32 images at a time.

The code for the final project was written in the Python programming language (https://www.python.org/). Python libraries were used to assist the coding process, including Numpy (http://www.numpy.org/), PyTorch (https://pytorch.org/), and Matplotlib (https://matplotlib.org/). The algorithm used was based on pretrained CNN. Training configurations such as learning rate, activation functions, and epoch settings were adjusted based on course material and online resources.

2.2 Transfer Learning Models

We used pretrained CNNs from the torchvision.models library:

- **ResNet-18**: We kept most of the layers frozen and fine-tuned the final block (layer4) to adapt the model to CIFAR-10. The fully connected layer was modified to produce output for 10 CIFAR-10 classes.
- **VGG-16**: We froze the convolutional part of the network and updated the final classification layer to match the 10 output categories.

2.3 Activation Functions

We experimented with two activation functions:

- **ReLU**: Simple and fast activation function that outputs the input value if it's positive and zero if it's not. It helps deep learning models train efficiently, but sometimes it can cause neurons to "die" if they only see negative inputs.
- **LeakyReLU**: Allows a small, non-zero gradient when the input is negative. The neuron still learns and stays active even when the input is negative (Ralla, 2020).

2.4 Hyperparameter Tuning

Our experimental design included tuning of hyperparameters to evaluate their effects on model performance:

- **Learning Rates**: We tested two learning rates, 0.001 and 0.0001. A higher learning rate allows for faster convergence but might compromise instability, while a lower learning rate provides more stable learning at the cost of speed.
- **Training Epochs**: Models were trained for either 10 or 20 epochs. Longer training allows models to better learn patterns but may increase the risk of overfitting.

2.5 Optimization and Loss Function

All experiments used Stochastic Gradient Descent (SGD) with momentum and weight decay for optimization:

- **Momentum**: We set momentum to 0.9 to help the model move more smoothly through the loss. Momentum helps carry the optimizer in the direction, which can speed up training and improve convergence.
- **Weight Decay**: A small weight decay of 5e-4 was applied to prevent the model from fitting too closely to the training data. This serves as regularization, helping to reduce overfitting by slightly penalizing large weights.

We used CrossEntropyLoss as the loss function, which is suited for classification tasks like CIFAR-10.

2.6 Evaluation on Test Data

After final training, each model was evaluated on the separate 10,000-image CIFAR-10 test set. We reported:

- **Overall Accuracy**: The percentage of correctly predicted test samples.
- **Training and Validation Loss Curves**: Shows learning progression and helps identify overfitting or underfitting.
- **Precision, Recall, and F1-Score:** Provides overview of prediction quality
- **Class-wise Accuracy**: Analyzes to determine if the model performed better on some categories than others.
- **Qualitative Analysis**: Visualized sample prediction vs. ground truth

3 Experiment Results

3.1 Performance Summary

Model	Epochs	LR	Activation	Train Loss	Val Loss	Accuracy	Precision	Recall	F1-score
ResNet-18	10	0.001	ReLU	1.1419	0.982	64.86 %	0.65	0.65	0.65
ResNet-18	20	0.001	ReLU	0.8812	0.9161	71.23 %	0.71	0.71	0.71
ResNet-18	10	0.0001	ReLU	0.7314	0.8627	74.85 %	0.75	0.75	0.75
ResNet-18	20	0.0001	ReLU	0.6962	0.8683	75.65 %	0.76	0.76	0.76
VGG-16	10	0.001	ReLU	1.1871	1.0516	61.35 %	0.61	0.61	0.61
VGG-16	20	0.001	ReLU	1.0075	0.9959	65.88 %	0.66	0.66	0.66
VGG-16	10	0.0001	ReLU	0.9109	0.9538	67.78 %	0.68	0.68	0.68
VGG-16	20	0.0001	ReLU	0.8811	0.9573	68.67 %	0.69	0.69	0.69
ResNet-18	10	0.001	LeakyReLU	1.1374	0.9825	65.10 %	0.65	0.65	0.65

ResNet-18	20	0.001	LeakyReLU	0.8816	0.9117	70.89%	0.71	0.71	0.71
ResNet-18	10	0.0001	LeakyReLU	0.7296	0.8783	74.89%	0.75	0.75	0.75
ResNet-18	20	0.0001	LeakyReLU	0.6995	0.8676	75.60%	0.76	0.76	0.76
VGG-16	10	0.001	LeakyReLU	1.1893	1.0614	61.45%	0.61	0.61	0.61
VGG-16	20	0.001	LeakyReLU	1.0069	0.9977	65.47%	0.65	0.65	0.65
VGG-16	10	0.0001	LeakyReLU	0.9081	0.9669	68.10%	0.68	0.68	0.68
VGG-16	20	0.0001	LeakyReLU	0.8802	0.9632	68.52%	0.68	0.69	0.68

Table 1. Overall results across all models

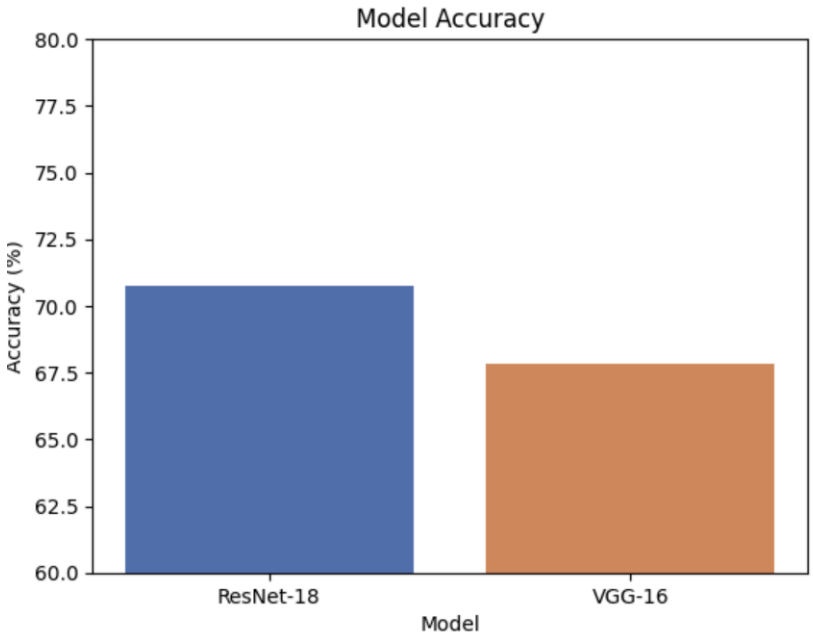

Figure 1. Comparison of average classification accuracy between ResNet-18 and VGG-16 models across all training configurations.

As shown in Figure 1, ResNet-18 consistently performed better than VGG-16 in test accuracy. This is likely due to its deeper structure and the inclusion of residual connections, which help

stabilize gradient flow during training. The best overall performance was achieved by ResNet-18 trained for 20 epochs with a learning rate of 0.0001 and ReLU activation, reaching an accuracy of 75.65% (Table 1). On average, VGG-16 performed worse than ResNet-18 by about 4–5% points in accuracy, which is consistent with expectation that ResNet extract features more effectively particularly with datasets like CIFAR-10. VGG-16 showed better performance when paired with LeakyReLU, especially at lower learning rates, indicating that this activation function helped reduce issues related to vanishing gradients in its deeper layers. Comparing both models, increasing the number of training epochs from 10 to 20 consistently improved results across all metrics. Performance gains accuracy and F1-score were common with longer training, showing the benefits of extended training time. Learning rate also played a role. While a higher rate (0.001) led to faster convergence, it also increased the risk of unstable learning. The lower learning rate (0.0001) provided more stable improvement, especially when paired with longer training. Overall, these results demonstrate the effect of tuning combination of model, activation function, learning rate, and epoch on CNN performance on classification tasks like CIFAR-10.

3.2 Loss Curves

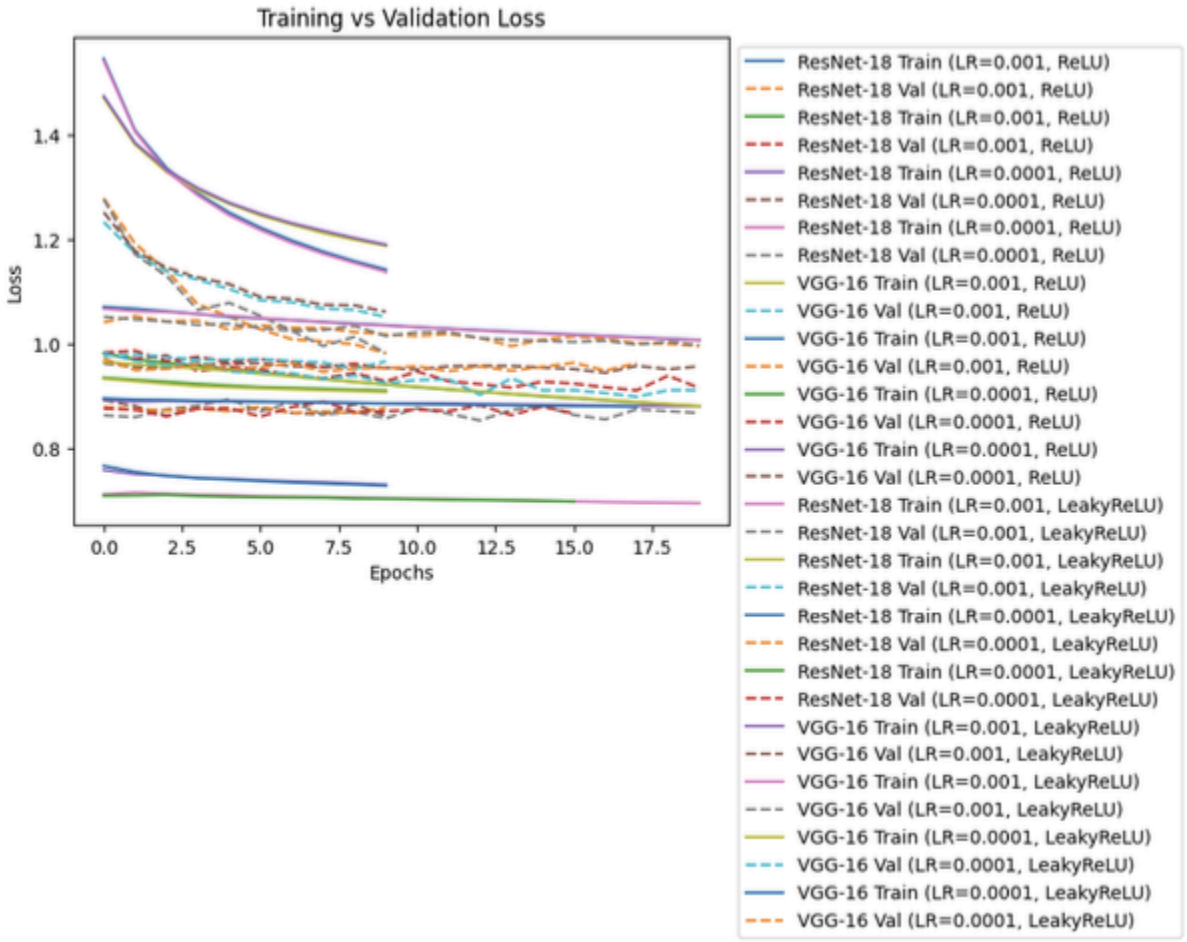

Figure 2. Training vs. validation loss curves across all model configurations.

Training and validation loss curves showed a downward trend, indicating that the models were learning over time. ResNet-18 showed faster and smoother convergence compared to VGG-16. We also noticed that models trained for 20 epochs tended to reach lower validation loss values than those trained for just 10 epochs, indicating that extended training allowed for better generalization. When using ReLU and a learning rate of 0.001, ResNet-18 achieved the lowest overall losses, suggesting that this combination was better performed. In contrast, models using LeakyReLU showed slightly higher losses, especially in ResNet-18, which may indicate that ReLU was a better fit for it. However, LeakyReLU helped improve stability in VGG-16, especially at lower learning rates.

3.3 Activation Function Analysis

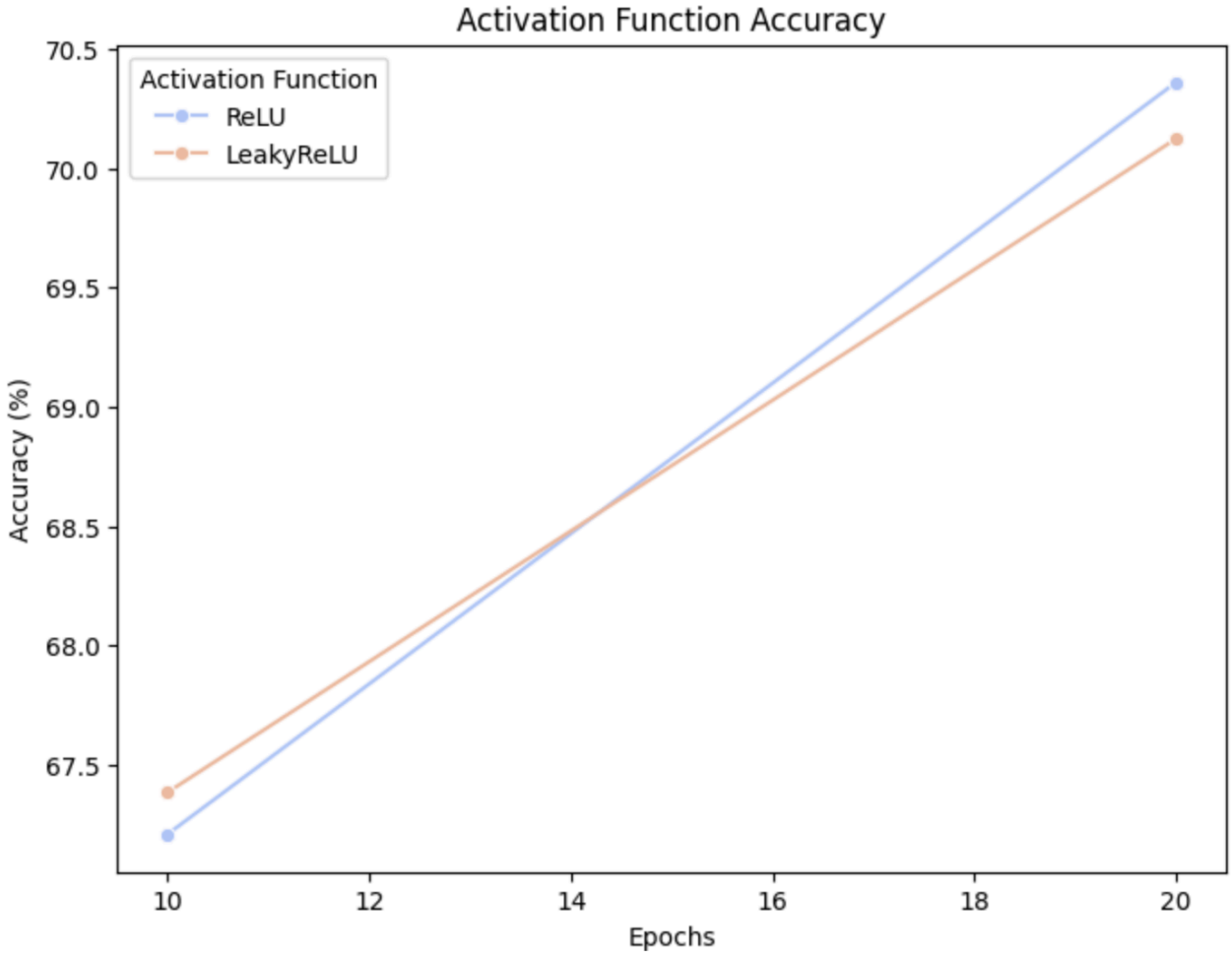

Figure 3. Activation accuracy comparing ReLu vs. LeakyRelu

When comparing activation functions, we found that LeakyReLU performed slightly better than ReLU at 10 epochs. However, ReLU outperformed LeakyReLU at 20 epochs, achieving higher final accuracy in both ResNet-18 and VGG-16 models. This indicates that while LeakyReLU

may offer an advantage in short training duration, ReLU leads to better long-term performance, especially when paired with deeper models and longer training.

3.4 Learning Rate

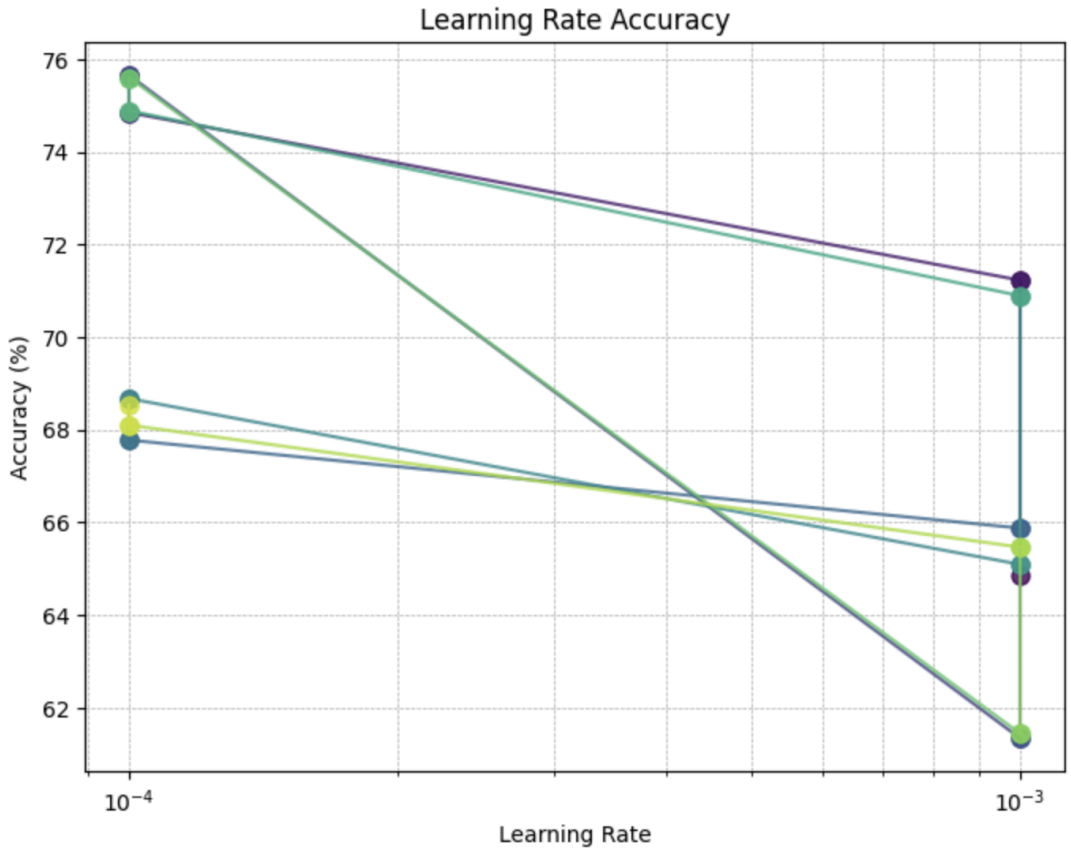

Figure 4. Comparison of test accuracy between learning rates 0.001 and 0.0001 across all configurations

When comparing learning rates, we saw that lower values (0.0001) consistently led to more stable training across both ResNet-18 and VGG-16. Although these models required longer training (20 epochs) to reach optimal performance, the slower convergence allowed them to generalize better. In contrast, models trained with a higher learning rate (0.001) converged more quickly, but their performance was not as good. The performance gap between the two learning rates narrowed in VGG-16.

3.5 Class-wise Performance Analysis

Class	ResNet-18 Accuracy (%)	VGG-16 Accuracy (%)
Airplane	72.3	69.7
Car	78.9	81.9
Bird	65.2	58.0
Cat	52.1	45.8
Deer	64.8	57.2
Dog	66.9	63.0
Frog	82.6	82.1
Horse	74.5	71.3
Ship	79.1	78.0
Truck	71.2	71.1

Table 2. Class-wise accuracy comparison between ResNet-18 and VGG-16

To better understand how each model performed across different categories, we reported class-wise accuracy for both ResNet-18 and VGG-16. ResNet-18 had the highest per-class accuracy on the frog class (82.60%), followed by ship (79.10%) and car (78.90%). It struggled most with cat (52.10%) and bird (65.20%). VGG-16 also showed strong performance on frog (82.10%) and car (81.90%), but had the lowest accuracy on cat (45.80%) and deer (57.20%). These results show that both models perform well when class boundaries are visually different but tend to confuse animal classes with more similar shapes and textures. Overall, ResNet-18 outperformed VGG-16 in most categories.

4. Conclusion

In this study, we explored how two pretrained CNN models—ResNet-18 and VGG-16—can be adapted for classifying images from the CIFAR-10 dataset. By applying data augmentation and tuning hyperparameters, we found that tuning these models can lead to varying performance.

Our experiments showed that ResNet-18 consistently outperformed VGG-16, reaching a peak accuracy when using the ReLU activation function. We found that a learning rate of 0.001 combined with 20 epochs of training provided the most promising results, showing the importance of a balanced and stable approach. While the activation function choice did influence outcomes, the difference between ReLU and LeakyReLU was minimal, with ReLU having a slight advantage. Additionally, our class-wise evaluation revealed that some animal categories are more challenging, suggesting that further refinements and more refined augmentation could be beneficial.

Future research could explore advanced data augmentation techniques, other optimizers like Adam, or custom network architectures. Expanding this analysis to other datasets or more complex tasks could offer deeper insights into the performance of these models.

5. References

Activation Functions: ReLU vs Leaky ReLU. Available at: https://medium.com/@sreeku.ralla/activation-functions-relu-vs-leaky-relu-b8272dc0b1be

CIFAR-10 Dataset. Available at: https://www.tensorflow.org/datasets/catalog/cifar10

COGS 181 Course Materials. Homework 4 and 5. University of California, San Diego, Winter 2025.

Jupyter Project. Available at: https://jupyter.org/

Matplotlib Documentation. Available at: https://matplotlib.org/

NumPy Documentation. Available at: http://www.numpy.org/

PyTorch CIFAR-10 Transfer Learning Tutorial. Available at: https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html

PyTorch Documentation. Available at: https://pytorch.org/