Assignment 2

Student: INNOCENT KISOKA; Student's email: kisoki@usi.ch

November 4, 2024

Image classification using CNNs

Question 2: In this section, I loaded and inspected the CIFAR-10 dataset, which consists of 60,000 32×32 color images across 10 classes, such as airplanes, automobiles, and animals.

0.1 Implementation

Loading: Using `torchvision.datasets.CIFAR10`, I downloaded and converted the images to tensors using `transforms.ToTensor`. This step converts image pixel values from [0, 255] to a [0, 1] range, which is generally more suitable for neural networks, as smaller values allow gradients to propagate more effectively during backpropagation.

Visualization: To gain insights into the dataset, I displayed one representative image per class. Visualizing samples helps verify the integrity of the data and ensures that transformations are applied correctly. Below figure is the output

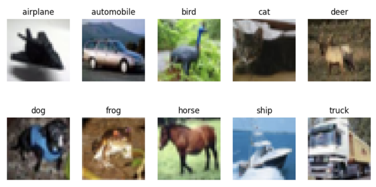


Figure 1: image per each class

Distribution Analysis: I plotted histograms of the class distributions in the training and test sets. This helps identify whether certain classes dominate the dataset, which could introduce bias in model predictions.

Observations

- The dataset has balanced classes with roughly equal samples for each category, reducing the risk of class bias during training.

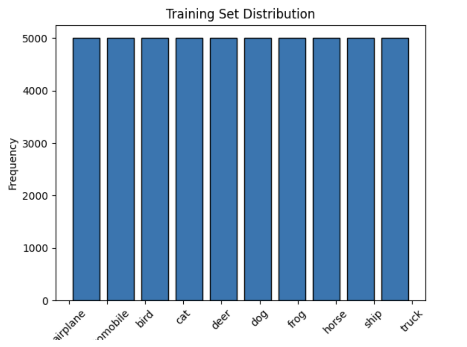


Figure 2: Histogram of distribution training set

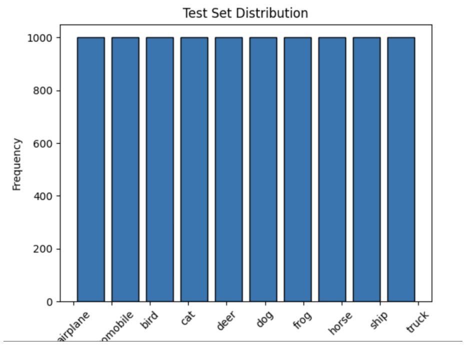


Figure 3: Histogram of distribution test set

- The images display correctly, confirming that the `ToTensor` transformation was applied accurately.

Question 3: Are the Entries in the Correct Type for the DL Framework in PyTorch?

In PyTorch, data needs to be in `torch.Tensor` format for compatibility with deep learning models. When using `torchvision.datasets.CIFAR10` to load the CIFAR-10 dataset, each entry initially consists of:

- An image in `PIL.Image` format, which is not directly usable by PyTorch models.
- A label as an integer, which is compatible with PyTorch for classification tasks.

So, while the label is in the correct type, the image needs to be converted from `PIL.Image` to `torch.Tensor` for compatibility with PyTorch's models.

How Can You Arrive at a Suitable Format for Your Training Pipeline?

To convert each entry to a suitable format for PyTorch, we can use `torchvision.transforms`, specifically `transforms.ToTensor`. This transformation:

- Converts `PIL.Image` objects into `torch.Tensor` format.
- Scales the pixel values from a range of [0, 255] (common in images) to a range of [0, 1], which helps stabilize and speed up training.

Explanation:

- **Element Type:** Each entry in CIFAR-10 is a tuple containing an image and its label. The image is a `torch.Tensor` and the label is an integer, both of which are ideal for use in PyTorch.
- **Tensor Conversion:** Since `ToTensor` is applied in the transformation, images are already in `torch.Tensor` format. This format makes further conversions unnecessary, simplifying the pipeline.
- **Image Dimensions:** Each image tensor has dimensions (3, 32, 32):
  - 3 refers to the three color channels (Red, Green, Blue),
  - 32 and 32 are the image's height and width in pixels.
- **Dimensional Importance:** Maintaining the (Channels, Height, Width) order ensures compatibility with PyTorch's `Conv2D` layers, which expect this format for processing images.

Question 4: Normalizing the Dataset

Normalization scales features to a standard range, which aids model training by stabilizing gradients and accelerating convergence.

The normalization values for CIFAR-10, mean = (0.4914, 0.4822, 0.4465) and std = (0.247, 0.243, 0.261)are standard values computed from the dataset. These values represent the mean and standard deviation for each color channel (Red, Green, Blue) across the entire CIFAR-10 training set.

Question 5: Validation Set for Model Monitoring

A validation set is crucial for monitoring model performance and tuning hyperparameters to prevent overfitting by assessing the model on unseen data.

Explanation

- **Role of Validation:** The validation set acts as a stand-in for real-world data, enabling an evaluation of the model's generalization. Observing validation accuracy and loss helps detect overfitting, where validation performance worsens while training accuracy improves.
- **Test Set Splitting:** Given that CIFAR-10 includes only training and test sets, I split the test set, allocating 20% for validation and 80% as a final test set.

Question 6: In this section , I managed to get the test accuracy of 51.98%

Architecture Choice: The model follows a Conv-Conv-Activation-Pool structure, repeated twice, followed by a fully connected layer. This architecture is effective for hierarchical feature extraction in images.

Layer Choices: Convolutional layers use a kernel size of 3, padding of 0, and stride of 1 to capture fine-grained features. Max pooling layers reduce spatial dimensions, focusing information for subsequent fully connected layers.

Output Layer: The final fully connected layer has 10 output units, aligning with the 10 classes in CIFAR-10.

Question 7:

Model Design Summary

Learning Rate

- Chosen rate: 0.03
- Balances training speed and optimization stability
- Allows rapid parameter updates without overshooting
- Higher rates might cause instability; lower rates could slow convergence

Epochs

- Number of epochs: 4
- Suitable for CIFAR-10's complexity
- Balances parameter optimization and overfitting risk
- Considers computational efficiency and time constraints
- Based on validation accuracy monitoring

Model Architecture

- Designed for balance between feature extraction and efficiency
- Incorporates multiple convolutional layers for hierarchical learning

- Uses carefully chosen filter numbers for representational capacity
- Includes pooling layers for spatial invariance
- Has fully connected layers for complex decision boundaries
- I Used ReLU activations to introduce non-linearity
- Considers potential regularization techniques

The architecture aims to provide a strong baseline for CIFAR-10 classification while avoiding overfitting and maintaining reasonable training times.

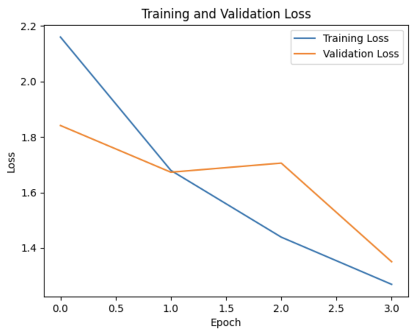


Figure 4: Training and Validation losses plot

Question 8: Analysis of Training and Validation Loss Trends

(a) Both training and validation losses decrease consistently over epochs, indicating effective learning.

(b) Training loss decreases more rapidly than validation loss initially, but they converge towards the end, suggesting good generalization.

(c) In the final epoch, both losses continue to decrease, indicating potential for further training without immediate risk of overfitting.

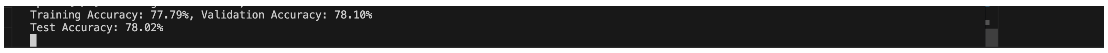
(d) No signs of overfitting are observed, as the validation loss doesn't show an upward trend and closely follows the training loss.

(e) The model appears to be learning and generalizing well, maintaining similar performance on both training and validation data.

Overall, I observed a well-behaved learning process with good generalization and no immediate concerns of overfitting.

Question 9: Justification of Choices (Summary)

I managed to acquire a test accuracy of 78.02 %



```
Training Accuracy: 77.79%, Validation Accuracy: 78.10%
Test Accuracy: 78.02%
```

Figure 5: Improved convnet results

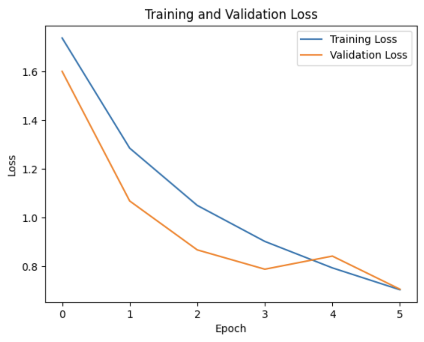


Figure 6: Improved Convnet Plot for training and validation losses

- **Learning Rate (0.001):** Chosen to ensure balanced and steady convergence without overshooting or slowing down excessively, allowing the model to improve accuracy while minimizing divergence risk.
- **Weight Decay (1e-4):** Applied as L2 regularization to reduce overfitting by penalizing large weights, helping the model generalize better on unseen data.
- **Dropout (0.5):** Used after fully connected layers to prevent overreliance on specific neurons, promoting robust learning and reducing overfitting.
- **Increased Epochs (6):** Allows more training time to counter the slight convergence slowdown from regularization, enabling the model to reach high accuracy.

These choices optimize accuracy and generalization, enhancing performance on both training and test sets.

Question 10: After training with different seeds, I observed the following:

- **Variability in Test Accuracy:** Each run showed a slight differences due to different weight initializations, revealing the model's sensitivity to initialization.
- **General Trend:** There were Minimal variation in test accuracy across seeds which indicates stable performance.
- **Mean Accuracy Convergence:** There was consistent test accuracies across runs.