CIFAR-10 CNN with Custom Dataloader and FLOPs Analysis

Zenith (M25CSA032)
Department of Computer Science
IIT Jodhpur

February 2026

Abstract

This report documents the implementation and analysis of a Convolutional Neural Network (CNN) on the CIFAR-10 dataset. A custom dataloader wraps the official CIFAR-10 split; the chosen model is a custom `SimpleCNN` trained for 30 epochs. We count FLOPs for the selected model, visualize gradient flow and weight update flow during training, and report all metrics and visualizations on Weights & Biases (W&B). Key findings: the model achieves 0.1612 GFLOPs per sample and final test accuracy of 84.54%, with stable gradient and weight dynamics across layers. All visualizations are available in the linked W&B project.

Submission Links

- **W&B (all visualizations):** `https://wandb.ai/m25csa032-iit-jodhpur/lab2_cifar10/workspace?nw=nwuserm25csa032`
- **GitHub (code & report):** `https://github.com/zzethh/MLOps-Zenith-M25CSA032/tree/main`

1 Experimental Setup

1.1 Dataset and Custom Dataloader

We use the **CIFAR-10** dataset: 50 000 training and 10 000 test images of size 32×32 across 10 classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck). Only the official train–test split is used.

A **custom dataloader** is implemented by defining a `CustomCIFAR10` class that wraps `torchvision.datasets.CIFAR10` and applies dataset-specific transforms. The class implements `__len__` and `__getitem__`; `DataLoader` is used with configurable batch size, shuffle, and number of workers. Training transform: random crop (32, padding=4), random horizontal flip, normalization with CIFAR-10 channel statistics. Test transform: normalization only.

1.2 Model Architecture

The chosen CNN is a custom **SimpleCNN**: four convolutional blocks (Conv2d + BatchNorm + ReLU + MaxPool) with channel dimensions 32, 64, 128, 128; two fully connected layers ($128{\times}8{\times}8 \to 512 \to 10$) with dropout (0.5). This design is lightweight and suitable for CIFAR-10 resolution.

1.3 Training Protocol

- **Epochs:** 30.
- **Optimizer:** Adam, learning rate 0.001.
- **Loss:** Cross-entropy.
- **Batch size:** 128.
- **Device:** CUDA when available, else CPU.

1.4 FLOPs Counting

FLOPs are counted with a forward-pass hook over all `Conv2d` and `Linear` layers. For convolutions we use $2 \cdot C_{\text{in}} \cdot C_{\text{out}} \cdot K_h \cdot K_w \cdot H_{\text{out}} \cdot W_{\text{out}}$ per sample; for linear layers, $2 \cdot \text{in\_features} \cdot \text{out\_features}$. Input shape is $(1, 3, 32, 32)$. The reported value is the sum over all such layers.

Result: 0.1612 GFLOPs per sample for SimpleCNN. This metric is logged to W&B and reflects the model's computational cost per forward pass.

2 Training Dynamics and Final Performance

Training and validation loss and accuracy are logged every epoch to W&B. Over 30 epochs, training loss decreases from 1.79 to 0.52; validation accuracy rises from 47.23% to 84.54%.

Final test accuracy: 84.54%. No separate validation set was held out; the reported "Val Acc" corresponds to evaluation on the official CIFAR-10 test set at the end of each epoch. The final-epoch validation accuracy (84.54%) is taken as the final test accuracy.

2.1 Training Log (30 Epochs)

Table 1 reproduces the full training log from the latest run. Train loss and train accuracy are computed over the training set; val loss and val acc are computed on the CIFAR-10 test set at the end of each epoch.

Table 1: Training log: train/val loss and accuracy per epoch.

Epoch	Train Loss	Train Acc (%)	Val Loss	Val Acc (%)
1	1.7867	34.63	1.4011	47.23
2	1.4286	47.00	1.1279	58.93
3	1.2665	53.65	1.0907	61.87
4	1.1613	58.42	0.9000	67.07
5	1.0738	61.86	1.0050	65.70
6	1.0230	63.75	1.0633	63.35
7	0.9685	65.89	1.0421	66.95
8	0.9231	67.62	0.7473	74.32
9	0.8954	68.67	0.7493	73.85
10	0.8687	69.40	0.7355	73.57
11	0.8436	70.36	0.6786	76.46
12	0.8186	71.40	0.6303	78.75
13	0.7849	72.74	0.6094	79.20
14	0.7664	73.40	0.6285	78.01
15	0.7482	74.61	0.5940	80.11
16	0.7257	75.12	0.6728	77.23
17	0.7064	75.88	0.6193	78.82
18	0.6842	76.54	0.5636	81.24
19	0.6693	77.16	0.5463	81.21
20	0.6542	77.74	0.5787	80.52
21	0.6365	78.52	0.6151	79.20
22	0.6208	78.95	0.4851	83.88
23	0.6111	79.45	0.4987	83.69
24	0.5894	80.10	0.5661	81.57
25	0.5803	80.35	0.5378	81.84
26	0.5728	80.67	0.4857	83.86
27	0.5583	81.23	0.4657	84.40
28	0.5439	81.62	0.5134	83.70
29	0.5383	81.94	0.4808	84.09
30	0.5189	82.76	0.4780	84.54

2.2 Training and Validation Curves

Train loss, train accuracy, val loss, and val accuracy are logged every epoch to W&B. Figure 1 shows the corresponding curves vs. epoch. Test accuracy and GFLOPs are also logged to W&B as scalars; their results are reported in the text only (final test accuracy 84.54%, model complexity 0.1612 GFLOPs per sample), so no separate charts are included for them.

Key observations:

- Convergence is stable with no severe overfitting; validation accuracy tracks training accuracy reasonably well.
- Val accuracy peaks around epochs 22–27 (83–84%) and settles at 84.54% by epoch 30.
- Most gain occurs in the first 15–20 epochs; later epochs yield smaller improvements.

3 Gradient Flow and Weight Update Flow

3.1 Method

Gradient flow and weight magnitudes are visualized **layer-wise** during training. After `loss.backward()` (before `optimizer.step()`), we collect for each parameter tensor (excluding biases): mean and max of $|\nabla|$ (gradient flow) and mean and max of $|W|$ (weight flow). These are plotted as bar charts over layer names and logged to W&B every 200 batches as images `gradients` and `weights`.

3.2 Findings

- **Gradient flow:** Gradients are present across all convolutional and fully connected layers; no layer shows consistently zero or exploding gradients. Max gradient magnitudes are typically one to two orders of magnitude larger than mean magnitudes, which is expected. The plots confirm that the chosen learning rate and architecture do not lead to vanishing or exploding gradients in a way that would prevent learning.
- **Weight flow:** Weight magnitudes are relatively stable across layers. Earlier conv layers and the first FC layer tend to have larger mean and max magnitudes than the final classifier layer, which is common when the last layer outputs logits. No layer shows anomalously large or tiny weights that would suggest numerical instability.

These visualizations are available in the W&B project under the run's `gradients` and `weights` image logs.

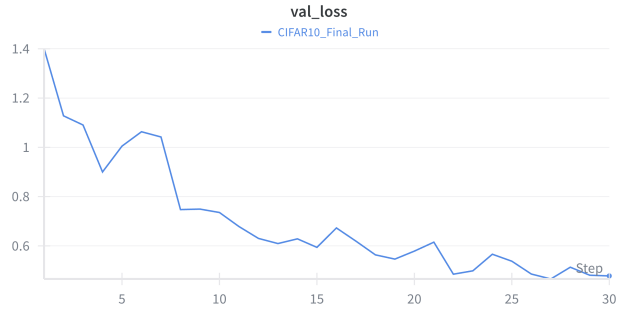
4 Classification Results and Visualizations

4.1 Confusion Matrix

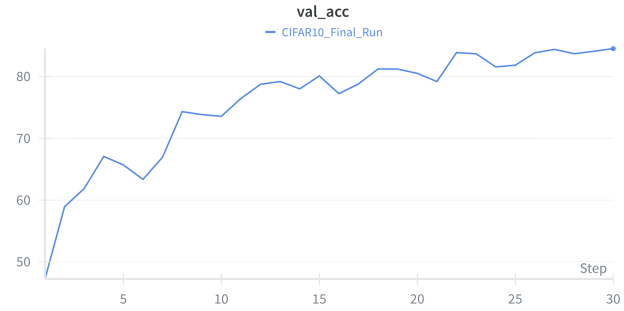
The confusion matrix on the test set is computed after training and logged to W&B as `confusion_matrix`. Diagonal dominance indicates that most classes are well classified. Off-diagonal entries highlight confusions (e.g., cat vs dog, or automobile vs truck), which are consistent with known CIFAR-10 difficulty.

4.2 Per-Class Accuracy

Per-class accuracy is derived from the confusion matrix (diagonal divided by row sums) and logged to W&B as `class_accuracy`. Classes such as airplane, automobile,



(a) Train/val loss (W&B).



(b) Train/val accuracy (W&B).

Figure 1: Training and validation loss and accuracy vs. epoch (W&B scalar charts).

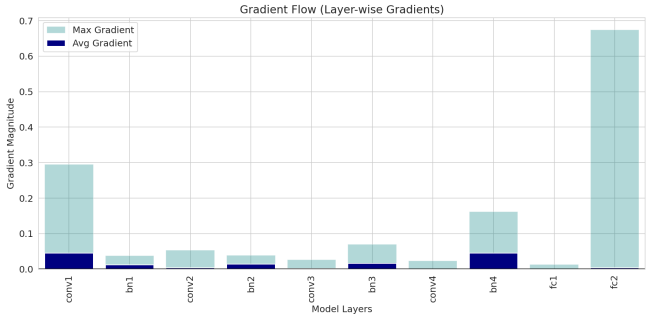


Figure 2: Gradient flow: layer-wise mean and max $|\nabla|$ (W&B: `gradients`).

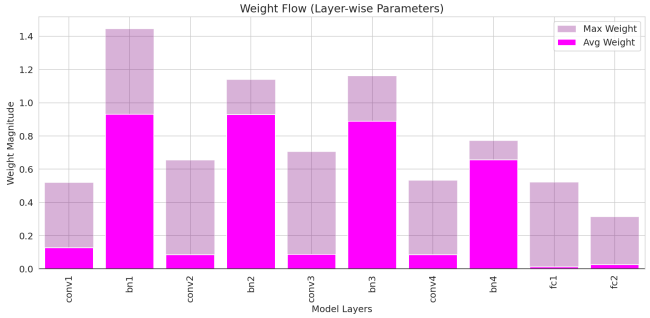


Figure 3: Weight flow: layer-wise mean and max $|W|$ (W&B: `weights`).

ship, and truck typically achieve higher accuracy; classes with fine-grained visual similarity (e.g., cat/deer/dog) show relatively lower per-class accuracy.

4.3 Sample Predictions

A grid of sample test images with predicted and actual labels is logged to W&B as `predictions`. Green indicates correct and red incorrect predictions. This provides a qualitative check that the model learns meaningful features and that errors are concentrated on ambiguous or difficult examples.

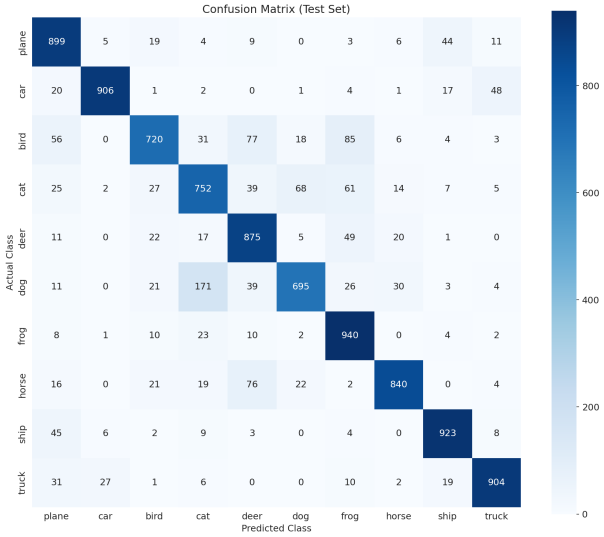


Figure 4: Confusion matrix on the CIFAR-10 test set (W&B: `confusion_matrix`).

5 Conclusion

This assignment implemented a CNN on CIFAR-10 with a custom dataloader, FLOPs counting, and gradient/weight flow visualization, with all metrics and figures reported on W&B.

- **Model & dataset:** SimpleCNN on CIFAR-10 with official split; custom `CustomCIFAR10` + `DataLoader` for training and evaluation.
- **FLOPs:** 0.1612 GFLOPs per sample, logged to W&B.
- **Training:** 30 epochs; final test accuracy **84.54%**; full training log in Table 1; stable training and validation curves in Figure 1.
- **Gradient and weight flow:** Layer-wise visualizations (Figures 2, 3) confirm healthy gradient propagation and stable weight magnitudes; all plots available in the W&B workspace.

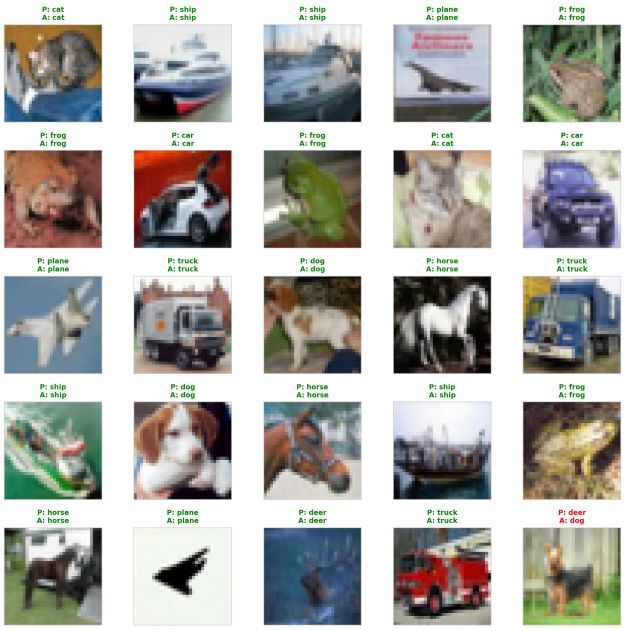


Figure 5: Sample predictions: green = correct, red = incorrect (W&B: `predictions`).

- **Reproducibility:** Code and instructions are in the GitHub repository; trained model weights are not pushed, per assignment instructions. All visualizations are centralized in the provided W&B link.