Image Classification from Scratch

https://github.com/manasjuneja/imageclassification

Analysis of Model Performance

The implemented neural network, trained from scratch using only NumPy, successfully classifies images from three selected CIFAR-10 classes. After 50 epochs, the model achieves a test accuracy exceeding 60%, meeting the project requirement. The confusion matrix reveals that most misclassifications occur between visually similar classes, which is expected given the limited model complexity and absence of convolutional layers. Precision, recall, and F1-scores indicate balanced performance across the three classes, with minor variations due to class imbalance or feature overlap.

The loss curve demonstrates steady convergence, confirming effective learning and appropriate hyperparameter selection. Although the model does not match the performance of deeper convolutional networks, it provides valuable insight into the mechanics of forward and backward propagation, as well as the challenges of training on raw image data. Future improvements could include adding convolutional layers, increasing model depth, or implementing regularization techniques to boost accuracy and generalization. Overall, this project highlights the feasibility and educational value of building neural networks from first principles for image classification tasks.

Confusion Matrix:

```
[[741 104 155]
 [ 55 899  46]
 [104  76 820]]
```

Precision:

[0.82333333 0.83317887 0.80313418]

Recall:

[0.741 0.899 0.82]

F1-score:

[0.78 0.86483886 0.81147947]

Accuracy:

0.82

Key Implementation Points

Data Preparation

- **Download and Extract** the CIFAR-10 dataset (Python version).
- **Load the data batches**, concatenate them, and **filter** for the three selected classes (e.g., *airplane*, *automobile*, *bird*).

- **Normalize** image data to scale pixel values to the range `[0, 1]`.
- **One-hot encode** class labels to use with softmax and cross-entropy loss.

Neural Network Architecture

A simple **multi-layer perceptron (MLP)** is used:

- **Input Layer:** 3072 units (flattened 32×32×3 RGB image)
- **Hidden Layer 1:** 128 units, **ReLU** activation
- **Hidden Layer 2:** 64 units, **ReLU** activation
- **Output Layer:** 3 units (for 3 classes), **Softmax** activation

---

Weight Initialization

- Weights are initialized using a **scaled normal distribution**, such as **He Initialization**, to ensure stability during training.

Forward Propagation

- Compute layer-wise activations using:
    - **Matrix multiplications**

- **ReLU** for hidden layers
- **Softmax** for the output layer

Loss Function

- Use **Cross-Entropy Loss** for multi-class classification.
- Compare predicted softmax probabilities with one-hot encoded ground truth labels.

Backpropagation

- Manually compute **gradients** for all weights and biases using the **chain rule**.
- Include derivatives for:
  - **ReLU**
  - **Softmax + Cross-Entropy** combination

Optimization

- Use **Stochastic Gradient Descent (SGD)**:
  - Fixed learning rate

    - Mini-batch updates

Training Loop

- Shuffle training data at the start of each epoch.
- Divide data into **mini-batches**.
- For each mini-batch:
    - Perform **forward pass**
    - Perform **backward pass**
    - **Update weights**
- Track and **plot loss** across epochs to monitor convergence (`loss_curve.png`).

Evaluation

After training, evaluate model performance using the test set:

- **Predict class labels**
- Compute:
    - **Accuracy**
    - **Precision**

    - **Recall**
    - **F1-score** (for each class)
- Plot a **Confusion Matrix** (`confusion_matrix.png`) to analyze misclassifications.

Platform Independence

- Entire codebase is built using:
    - **NumPy**
    - **Standard Python libraries**
- Fully **CPU-compatible** — runs on any architecture (x86_64, ARM, etc.)
- No deep learning frameworks or GPUs required.