UNIVERSITY OF NEBRASKA OMAHA

Computer Science

CSCI8590-001: Fundamentals Of Deep Learning

By: Dr. *Xin Zhong*

Assignment: 1

Submitted By: Zerin Shaima Meem

NUID: 77548102

c

UNIVERSITY OF Nebraska Omaha

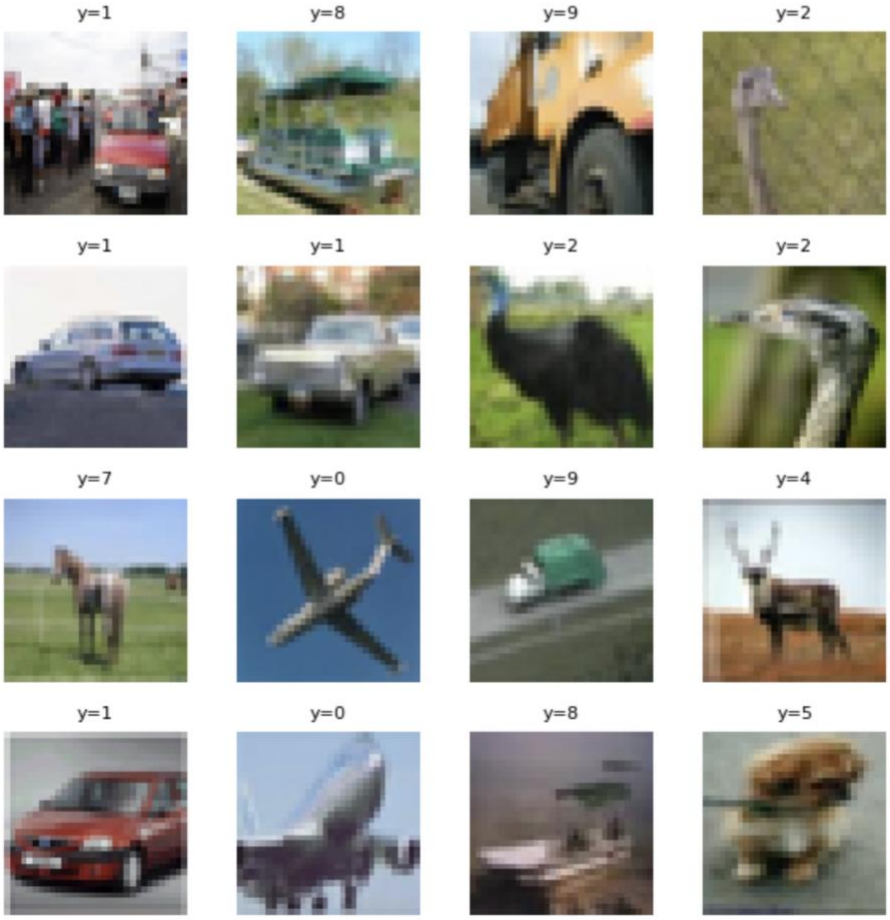
Runnable Code: Google Colab Link

Required Output:

- Displayed images

```
Dataset Info

x shape: (50000, 32, 32, 3)
x dtype: float32
x min/max: 0.0 1.0
y shape: (50000,)
y dtype: int64
unique classes: [0 1 2 3 4 5 6 7 8 9]
16 random images in a 4x4 grid.
```



- Printed dataset sizes for all split methods

```
=== Split method checks ===
Hold-out train size: 40000 val size: 10000
intersection =  0


K-fold train size: 40000 val size: 10000
intersection =  0


Bootstrap train size: 50000 val(OOB) size: 18313
unique train samples: 31687 (duplicates expected)
intersection =  0

Train arrays: (40000, 32, 32, 3) (40000,)
Val arrays:   (10000, 32, 32, 3) (10000,)
```

- Printed batch shapes from the generator

```
=== Mini-batch generator test ===
Batch 1: x=(64, 32, 32, 3), y=(64,), x_min=0.0000, x_max=1.0000
Batch 2: x=(64, 32, 32, 3), y=(64,), x_min=0.0000, x_max=1.0000
Batch 3: x=(64, 32, 32, 3), y=(64,), x_min=0.0000, x_max=1.0000
Batch 4: x=(64, 32, 32, 3), y=(64,), x_min=0.0000, x_max=1.0000
Batch 5: x=(64, 32, 32, 3), y=(64,), x_min=0.0000, x_max=1.0000

=== Special-case test (small dataset) ===
Batch 1: x=(64, 32, 32, 3), y=(64,), x_min=0.0000, x_max=1.0000
Batch 2: x=(64, 32, 32, 3), y=(64,), x_min=0.0000, x_max=1.0000
Second batch is still same size
```

Discussion:

In this assignment, we used the CIFAR-10 dataset to explore basic data preprocessing, splitting techniques, and mini-batch generation.

- The original images are stored as integer values in the range [0,255] so we first converted them to floating-point values and normalized them to the range [0,1]. This step is important because working with normalized values makes learning more stable and avoids issues caused by large numeric ranges.
- After normalization, we printed and checked the dataset shape, data type, and value range to ensure the preprocessing was done correctly.

- To get a better understanding of the dataset, 16 random images were displayed in a 4×4 grid. It helped us to visualize the verity of data with their normalized labels.
- The we implemented, three different data splitting methods: hold-out, k-fold, and bootstrap.
- In the hold-out method, the dataset was randomly shuffled and split into training and validation sets using an 80/20 ratio. This approach is simple and computationally efficient, but its main limitation is that performance depends on a single random split.
- k-fold cross-validation is a more reliable splitting. In this method, the dataset is divided into multiple folds, and one fold is used for validation while the remaining folds are used for training.
- Lastly, we implemented Bootstrap sampling by randomly sampling the training set with replacement. As a result, some samples appear multiple times in the training set, and some other are not selected in the training set. The samples that are not selected are treated as out-of-bag samples and used for validation.
- All splitting methods were implemented using indices instead of directly copying data arrays. This makes the implementation more memory-efficient and ensures that the original dataset remains unchanged.
- The last task was to implement a mini-batch generator using Python's **yield** statement. The generator produces batches sequentially using a pointer that tracks the current position in the dataset.
- For the case, where the remaining samples are not enough to form a full batch, we reshuffled the dataset and reset the pointer. This ensures that all batches have a fixed size and that the data order changes between epochs.

Overall, this assignment helped us clarify the practical differences between common data splitting strategies and demonstrated how mini-batch generation can be implemented efficiently. This is very helpful for building reliable and reproducible machine learning experiments.