Advanced Topics in Neural Networks

Repede Monica-Gabriela

December 23, 2025

1 Introduction

In this project, we implement a small neural network that can approximate a fixed sequence of image transformations: resizing, grayscale conversion, horizontal and vertical flip, while trying to achieve faster inference than the corresponding sequential CPU implementation. The model takes as input RGB images of size $3 \times 32 \times 32$ and outputs grayscale images of size $1 \times 28 \times 28$.

The objective is not to perfectly reproduce the transformation, but to learn a sufficiently accurate approximation that minimizes the loss function.

2 Implementation of the model

2.1 Model arhitecture

To approximate the target transformation, we chose a linear model with a single fully connected layer in order to keep the architecture minimal and computationally efficient. During the forward pass, the input image is reshaped into a one-dimensional vector, passed through the linear layer, and then reshaped back into a grayscale image of size 1×28×28. This design intentionally avoids convolutional layers in order to keep the model as simple as possible.

The creative aspect of this approach lies in approximating the entire sequence of deterministic image transformations as a single learned linear mapping. Instead of applying resizing, grayscale conversion and flipping sequentially on the CPU, the model performs them in one step through a matrix multiplication.

2.2 Loss function

The Mean Squared Error (MSE) loss is used to train the model, as the task can be formulated as a regression problem where the goal is to predict continuous grayscale pixel intensities. MSE measures the average squared difference between the predicted and ground truth pixel values, encouraging the model to minimize large errors.

Since the objective of this project is not to perfectly reproduce the transformation but to approximate it while minimizing the overall error, MSE is a suitable and simple choice.

2.3 Early stopping

Early stopping is used as a stopping criterion to prevent unnecessary training once the model has converged. During training, the validation loss is monitored and training is terminated when no significant improvement is observed for several consecutive epochs.

This approach helps avoid overfitting to the training data and reduces overall training time. Since the model used in this project is simple and converges quickly, early stopping allows the training process to stop as soon as further optimization yields diminishing returns.

2.4 Train

We trained the model on 200 epochs, with a batch size of 128. We use a patience of 10 epochs and a difference between the best epoch and the current one of 0.0001. Also, Adam optimizer was used with a learning rate equal to 0.001. As it can be seen below, the training stopped after 32 epochs.

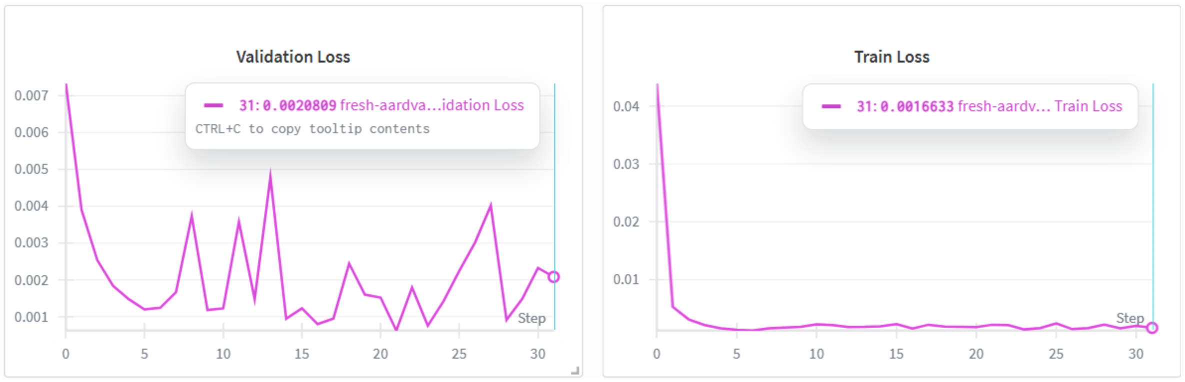

Figure 1: Train statistics

3 Comparing ground truth with the predicted images

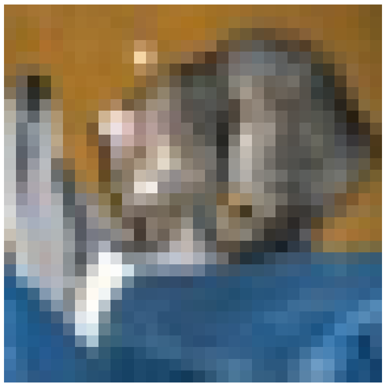

Figure 2: Input

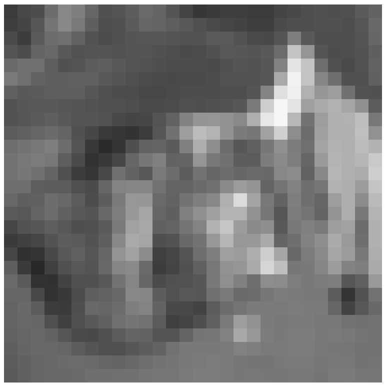

Figure 3: Ground truth

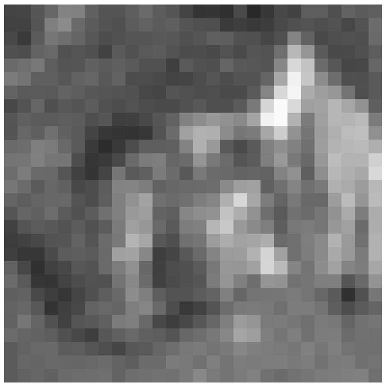

Figure 4: Predicted

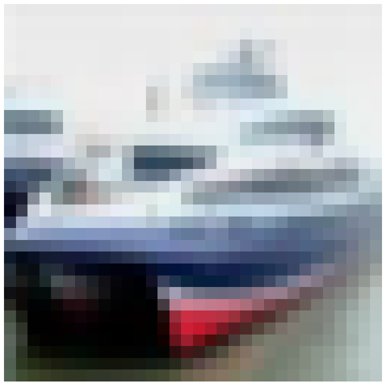

Figure 5: Input

Figure 6: Ground truth

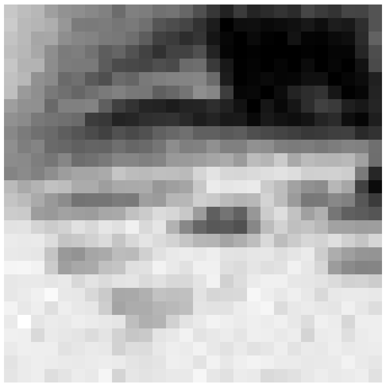

Figure 7: Predicted

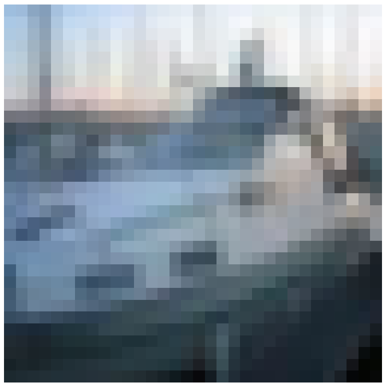
Figure 8: Input

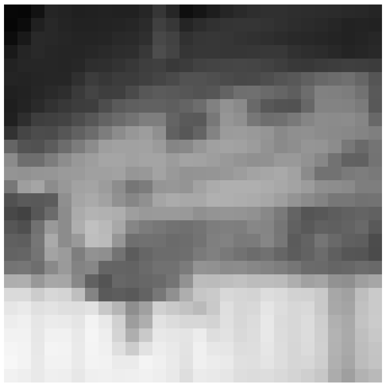
Figure 9: Ground truth

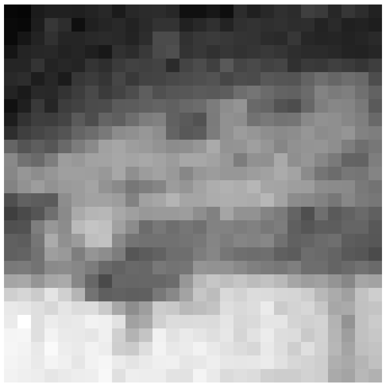
Figure 10: Predicted

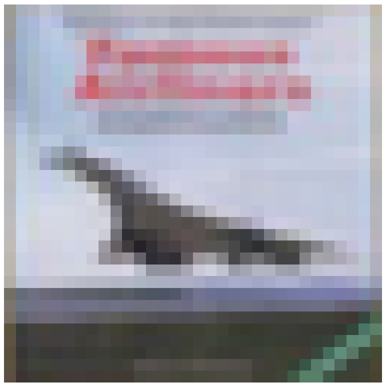
Figure 11: Input

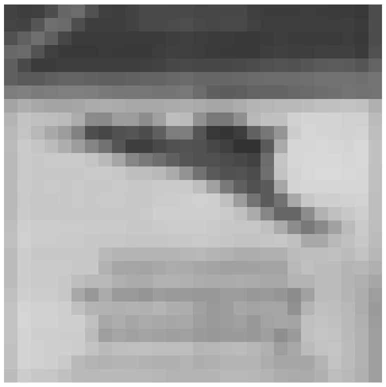
Figure 12: Ground truth

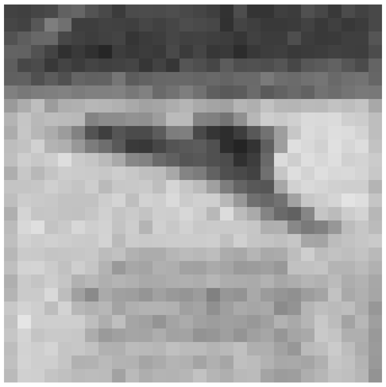
Figure 13: Predicted

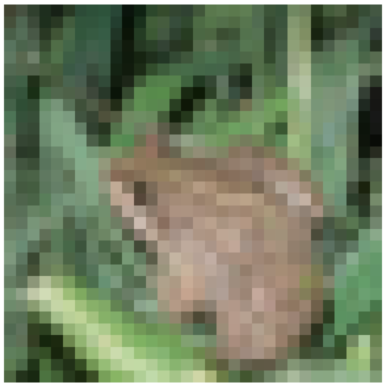
Figure 14: Input

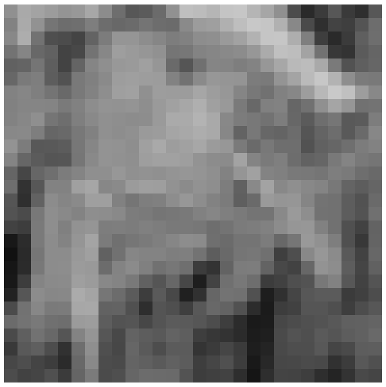
Figure 15: Ground truth

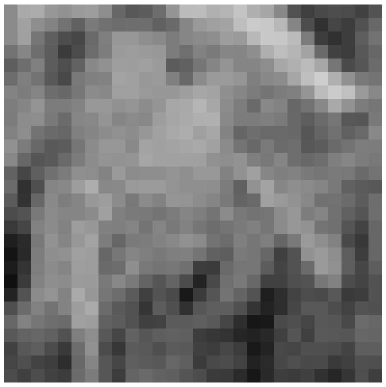
Figure 16: Predicted

4 Benchmark comparison

Device	Batch Size	Sequential Transforms	NN model Time
CPU	1024	1.134157332s	0.101427578s
CPU	512	1.133574057s	0.102554796s
CPU	256	1.125700002s	0.110185822s
CPU	128	1.119672969s	0.115661531s
CPU	64	1.146522188s	0.164159893s
CPU	32	1.128262938s	0.176648766s
CPU	16	1.12504965s	0.26585658s
CPU	8	1.154197367s	0.234170212s
GPU	1024	1.11670064s	0.302927236s
GPU	512	1.114984422s	0.32541693s
GPU	256	1.128414049s	0.29502032s
GPU	128	1.138344219s	0.324584602s
GPU	64	1.117484117s	0.362884111s
GPU	32	1.1229118s	0.496026868s
GPU	16	1.120999516s	0.704631729s
GPU	8	1.13008866s	1.160968829s

Table 1: Benchmark comparison with different parameters for batch size and device

From the table above, we can see that the model is faster than the sequential CPU transformations in almost all cases. On CPU, the model clearly outperforms the sequential approach for all tested batch sizes. On GPU, the model is faster for larger batch sizes, while for very small batches (such as batch size 8) the overhead of launching GPU operations makes it slower than the sequential implementation.

5 Conclusion

We showed that a minimal fully connected neural network can approximate a fixed image transformation pipeline and outperform a sequential CPU implementation under appropriate batching conditions. Although the model produces blurred outputs due to its limited inductive bias and regression loss, it satisfies the requirements of the task and demonstrates the potential of learned approximations for accelerating preprocessing pipelines.