Performance Benchmarking and Transformation Accuracy of Image Transformation on CPU vs. GPU

Marius Marin

December 4, 2023

Abstract

This report details the development and benchmarking of a convolutional neural network model designed to transform CIFAR-10 images into grayscale, resized, and flipped versions. The performance of sequential transformations on a CPU were compared with batch processing on both CPU and GPU. The model's accuracy in replicating ground truth transformations was also evaluated. Insights from Weights & Biases logs and visual comparisons are included to corroborate my findings.

1 Introduction

Efficient image processing is crucial in various applications of computer vision. This study focuses on benchmarking the performance of image transformation tasks on CPU and GPU, as well as the accuracy of these transformations using a custom PyTorch model on the CIFAR-10 dataset.

2 Model Architecture

The 'TransformationModel' was intentionally designed to be simple, composed of a single convolutional layer that introduces minimal learnable parameters. This design choice was motivated by the nature of the transformation task — which is fundamentally deterministic and does not require a deep or complex model architecture. The convolutional layer acts as a feature enhancer, allowing for a slight adaptation to the input data's peculiarities, which might not be captured by fixed operations alone.

3 Loss Function

The Mean Squared Error (MSE) loss function was selected for its propensity to penalize larger errors more severely than smaller ones, which is ideal for a task where precise pixel-wise accuracy is essential. MSE is a natural fit for regression tasks where the goal is to closely approximate a target value — in this case, the pixel values of the transformed images.

4 Early Stopping

Early stopping was utilized as a form of regularization to prevent overfitting. By monitoring the validation loss and stopping the training when the loss ceases to decrease, we ensure that the model generalizes well without unnecessarily extending the training process. This approach is particularly beneficial when computational resources are limited or when training larger models.

5 Benchmarking Methodology

We benchmarked the model's inference time using different batch sizes on both CPU and GPU (ensuring CUDA synchronization on the GPU). The aim was to determine the batch size at which the model's batch processing outperformed sequential transformations on the CPU. The device's computational capabilities were also taken into account.

6 Code Snippets

Below are key snippets from our codebase that illustrate the simplicity and effectiveness of our approach.

6.1 Model Definition

```python
class TransformationModel(nn.Module):
    def __init__(self):
        super(TransformationModel, self).__init__()
        self.conv = nn.Conv2d(3, 3, kernel_size=3, stride=1,
    padding=1)

    def forward(self, x):
        x = nnf.relu(self.conv(x))
        x = F.resize(x, size=(28, 28), antialias=True)
        x = F.rgb_to_grayscale(x, num_output_channels=1)
        x = F.hflip(x)
        x = F.vflip(x)
        return x
```

Listing 1: TransformationModel definition

6.2 Early Stopping Mechanism

```
class EarlyStopping:
    def __init__(self, patience=5, min_delta=0):
        self.patience = patience
        self.min_delta = min_delta
        self.best_loss = None
        self.patience_counter = 0

    def __call__(self, val_loss):
        if self.best_loss is None or val_loss < self.best_loss - self.min_delta:
            self.best_loss = val_loss
            self.patience_counter = 0
        else:
            self.patience_counter += 1
            if self.patience_counter >= self.patience:
                return True
        return False
```

Listing 2: EarlyStopping class

6.3 Loss Function Utilization

```
for images, ground_truth in train_loader:
    optimizer.zero_grad()
    images = images.to(device)
    ground_truth = ground_truth.to(device)
    outputs = model(images)
    loss = loss_function(outputs, ground_truth)
    loss.backward()
    optimizer.step()
    train_loss += loss.item()
```

Listing 3: Training loop snippet showing MSE loss calculation

6.4 Benchmarking Inference Time

```python
batch_sizes = [10, 50, 100, 200]
for batch_size in batch_sizes:
    sequential_time, model_batch_time = test_inference_time(model
    , device, batch_size)
    print(f"Sequential transforming time: {sequential_time:.4f}
   seconds")
    print(f"Model batch processing time: {model_batch_time:.4f}
   seconds")
```

Listing 4: Benchmarking inference time with varying batch sizes

7 Results

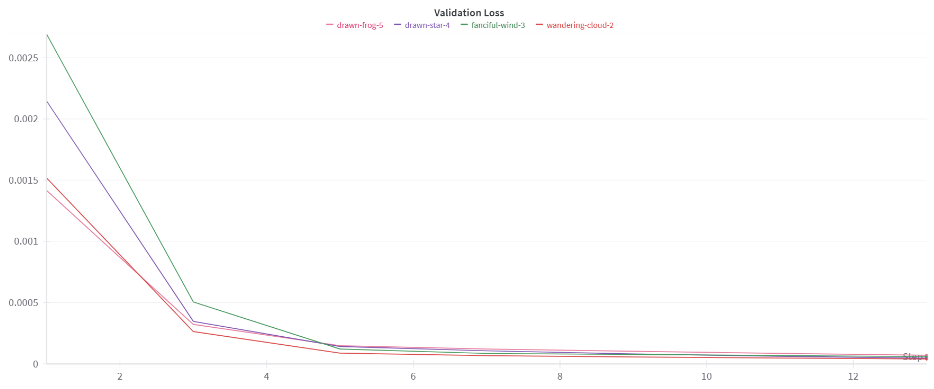

Figure 1: Validation Loss evolution on several runs

7.1 Transformation Accuracy

Consistent across all tests, the model achieved a Mean Squared Error (MSE) of 5.9×10^{-5}, showcasing its high precision in image transformation tasks. Visual inspections of the transformed images corroborated these findings, with the model-generated images displaying a close resemblance to their ground truth counterparts. This visual parity, despite the reduced dimensions and grayscale format, underscores the model's ability to capture and replicate the essential features of the input data.

7.2 Inference Time

Benchmarking across various environments and batch sizes yielded the following inference times, showcasing the model's enhanced processing speed:

- **PyCharm CPU**:
  - Batch size 10: Sequential time - 2.0077s; Model time - 2.0503s.
  - Batch size 50: Sequential time - 2.1514s; Model time - 1.5603s.
  - Batch size 100: Sequential time - 1.9046s; Model time - 1.6274s.
  - Batch size 200: Sequential time - 2.5341s; Model time - 1.6344s.
- **Colab CPU**:
  - Batch size 10: Sequential time - 2.7822s; Model time - 1.9597s.
  - Batch size 50: Sequential time - 3.2753s; Model time - 1.5744s.
  - Batch size 100: Sequential time - 2.4885s; Model time - 1.5125s.
  - Batch size 200: Sequential time - 3.4444s; Model time - 1.5173s.
- **Colab T4 GPU**:
  - Batch size 10: Sequential time - 3.1330s; Model time - 2.3993s.
  - Batch size 50: Sequential time - 3.0265s; Model time - 1.6880s.
  - Batch size 100: Sequential time - 3.5728s; Model time - 2.4954s.
  - Batch size 200: Sequential time - 2.9728s; Model time - 1.6241s.

These results showcase the advantages of batch processing using GPU acceleration, confirming that even with a straightforward convolutional neural network, it is possible to achieve both high accuracy and efficiency in image transformation tasks.

8 Image Comparisons

A selection of image comparisons is provided, which includes the model's output and the ground truth transformations.

Below are pairs of images showcasing the transformation accuracy of the model. On the left is the image processed by the model, and on the right is the corresponding ground truth image.

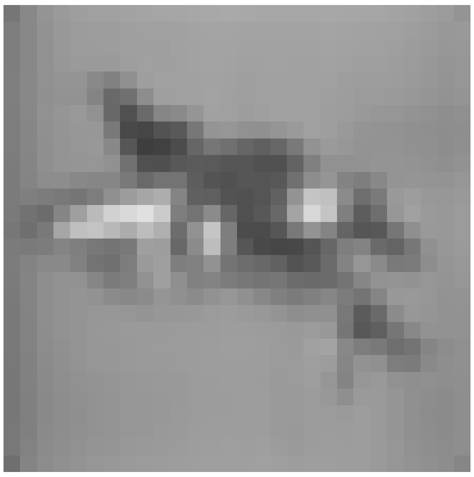

Figure 2: Processed by model

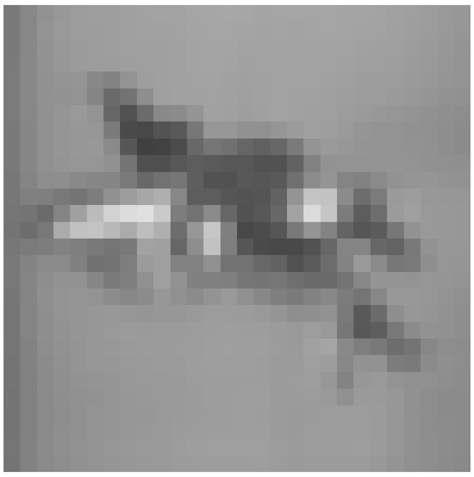

Figure 3: Ground truth

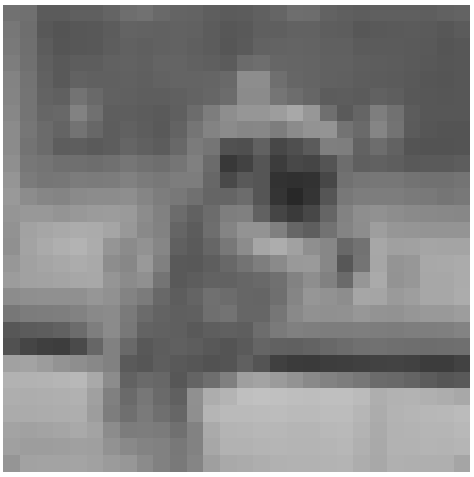

Figure 4: Processed by model

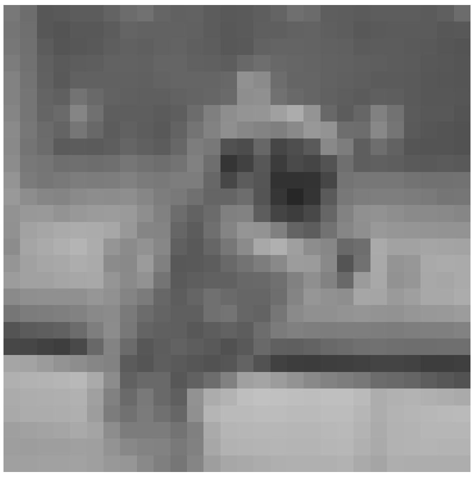

Figure 5: Ground truth

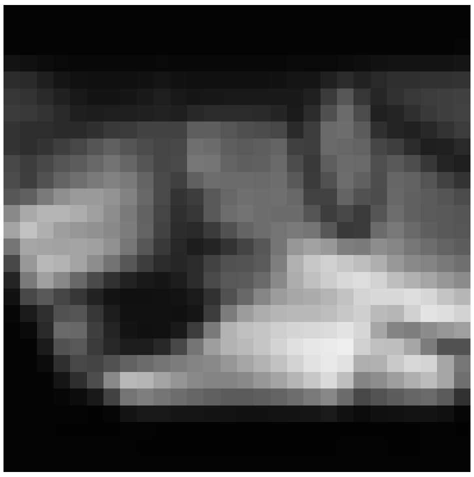

Figure 6: Processed by model

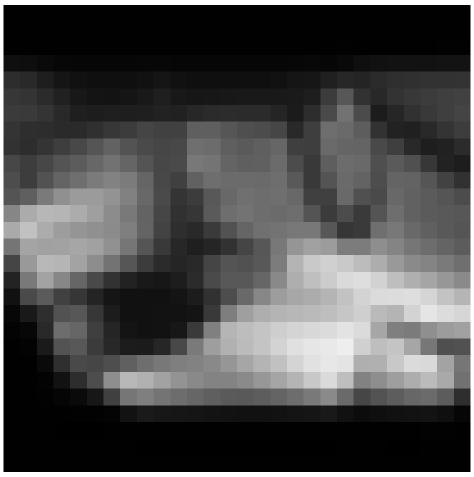

Figure 7: Ground truth

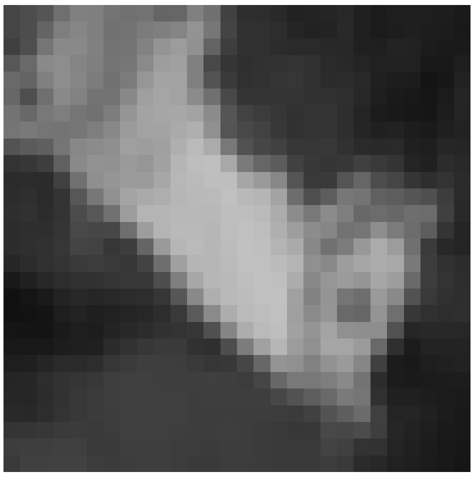

Figure 8: Processed by model

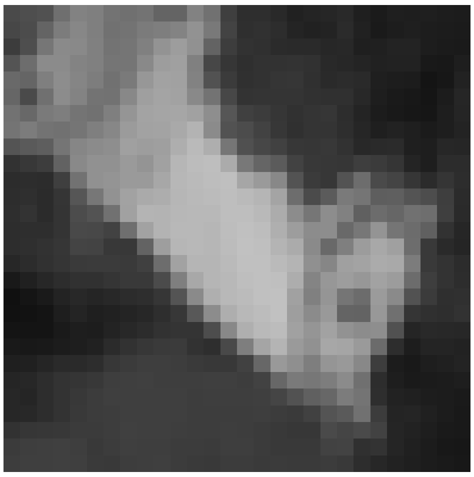

Figure 9: Ground truth

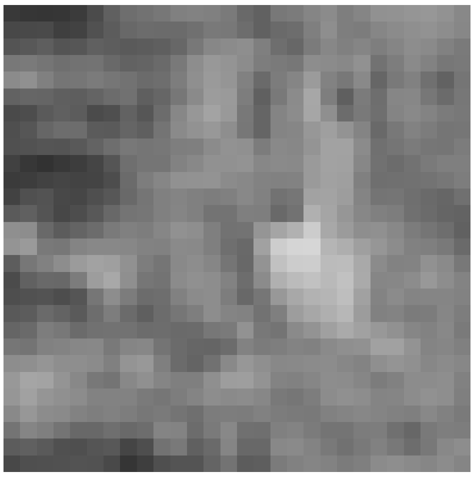
Figure 10: Processed by model

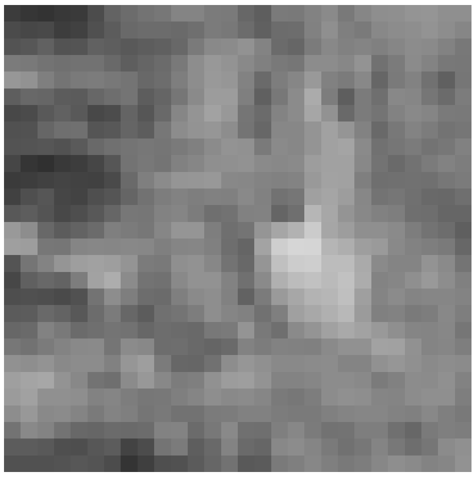
Figure 11: Ground truth

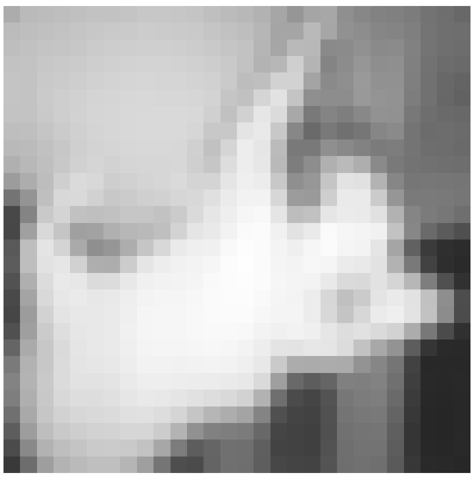
Figure 12: Processed by model

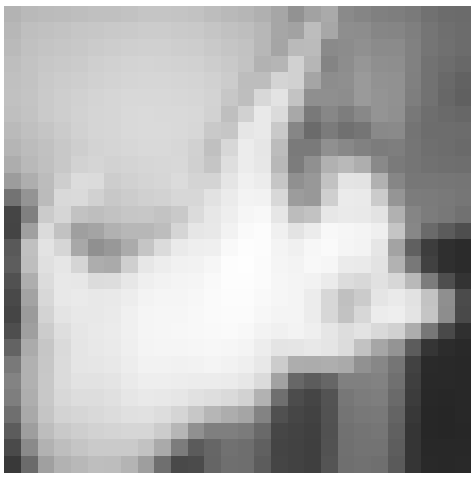
Figure 13: Ground truth

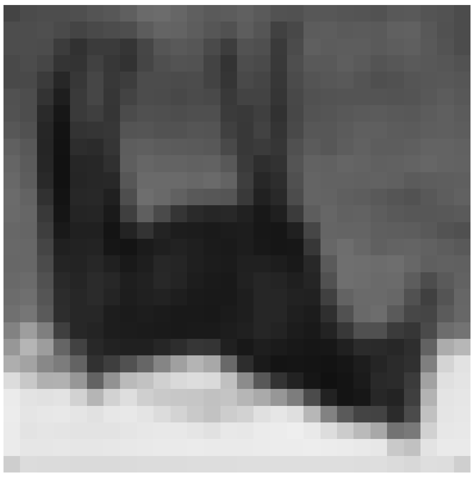

Figure 14: Processed by model

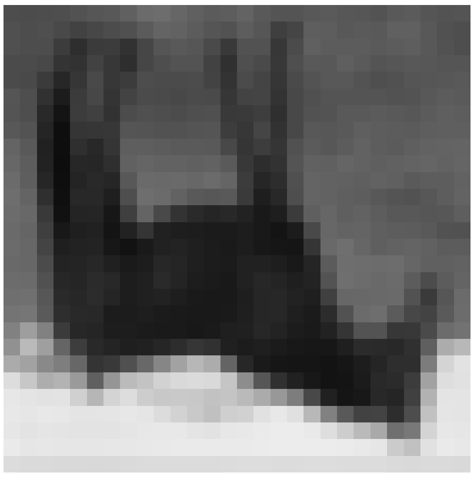

Figure 15: Ground truth

Figure 16: Processed by model

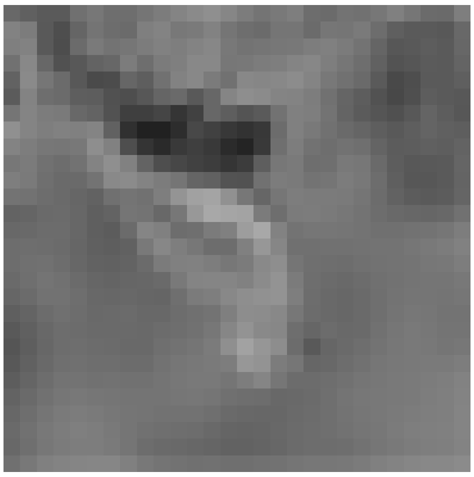

Figure 17: Ground truth

Figure 18: Processed by model

Figure 19: Ground truth

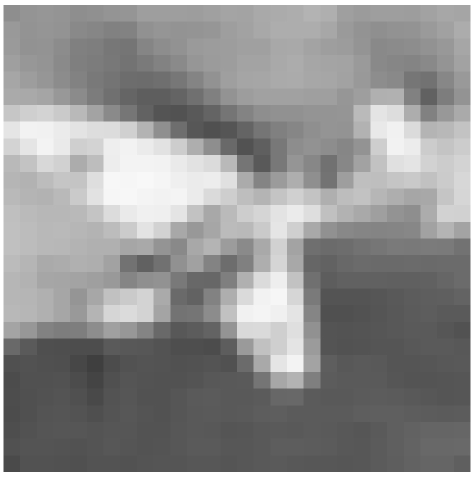

Figure 20: Processed by model

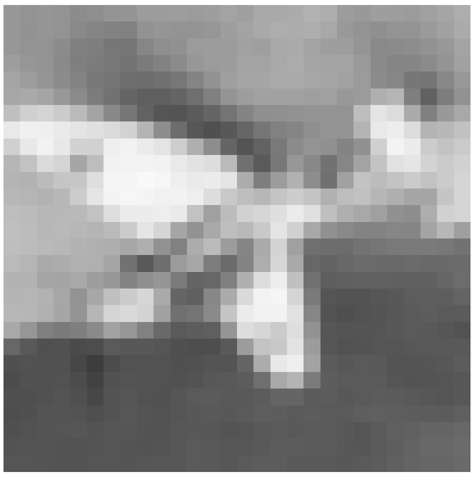

Figure 21: Ground truth

9 Conclusion

Batch processing, especially on GPUs, can greatly enhance the efficiency of image transformation tasks. The developed model serves as a testament to the potential of simple architectures in achieving high accuracy in such tasks.