Neural Network for Image Transformation

Pricop Tudor-Constantin

November 30, 2023

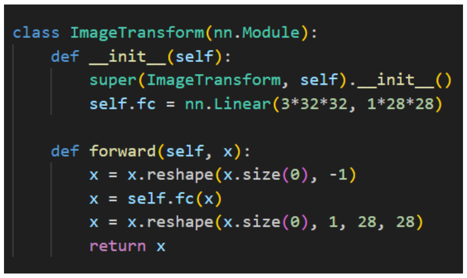
1 Introduction

This focus of this assignement revolves around building a neural network that imitates a combination of Pytorch transforms: resize, flip and grayscale, and optimizing it in order to outperform the classical transforms regarding time complexity.

2 Methods

My **neural network** is designed to learn the combined transforms of resizing, flipping horizontally and vertically, and grayscaling an image. As already mentioned in the task, the simplest model might work the best, thus the architecture of the network is as follows:

- Input Layer: Accepts an image of size $3 \times 32 \times 32$.
- Output Layer: Outputs the transformed image with size $1 \times 28 \times 28$.



```
class ImageTransform(nn.Module):
    def __init__(self):
        super(ImageTransform, self).__init__()
        self.fc = nn.Linear(3*32*32, 1*28*28)

    def forward(self, x):
        x = x.reshape(x.size(0), -1)
        x = self.fc(x)
        x = x.reshape(x.size(0), 1, 28, 28)
        return x
```

Figure 1: Neural network model.

The loss function chosen for the above method is **Mean Squared Error (MSE)**. MSE is a suitable loss function for generating transformed images due to its simplicity and effectiveness in measuring pixel-wise differences between the generated and target images. It calculates the average squared difference between corresponding pixel values, providing a direct and intuitive representation of the overall reconstruction error. When dealing with transformations like rotation, resizing or grayscale conversion, where preserving fine details and accurate pixel values is crucial, MSE encourages the model to minimize these differences and generate outputs that closely match the desired transformations. Additionally, MSE is well-suited for optimization algorithms, facilitating efficient convergence during training. While other loss functions may be employed based on specific use cases or priorities, MSE's straightforward nature and alignment with the goal of faithful image reconstruction make it a practical choice in many image transformation scenarios.

Early Stopping is a valuable tool in the training of neural networks models, as it helps prevent overfitting and optimizes the model's generalization performance. By monitoring the validation loss during the training process, the EarlyStopper allows training to cease if the validation loss fails to improve for a specified number of epochs, determined by the patience parameter. This early stopping mechanism safeguards against prolonged training that could lead to overfitting, where the model becomes excessively tailored to the training data and loses its ability to generalize well to unseen data. The delta parameter sets a minimum threshold for what constitutes a meaningful improvement, ensuring that only significant decreases in validation loss trigger the counter reset. In summary, the EarlyStopper contributes to more efficient training by halting the process once the model's performance on the validation set plateaus, thereby enhancing its ability to generalize to new data.

3 Results

After training our network on the CIFAR10 dataset, we observed satisfactory results as shown in the following tables...

Device	**Time for sequential transforms**	**Time for model**
CPU	2.15017	0.90675
GPU	1.86829	0.77130

Table 1: Comparison between time needed to do the transforms with Pytorch vs Model.

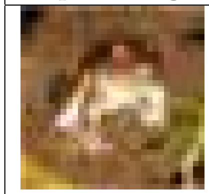
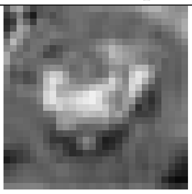
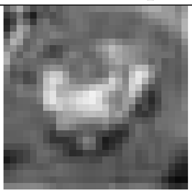
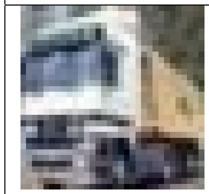
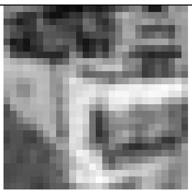
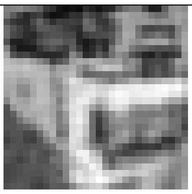
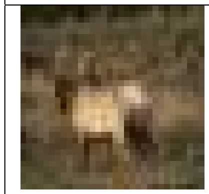
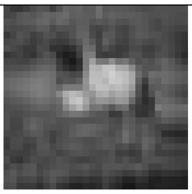
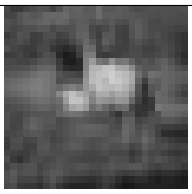
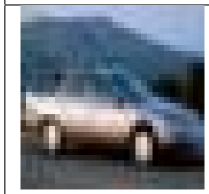
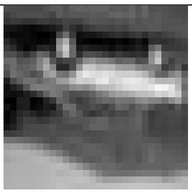
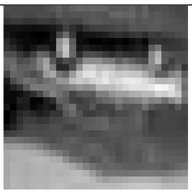
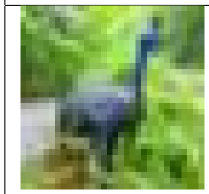
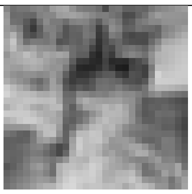
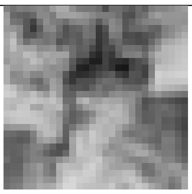
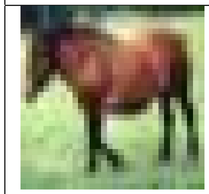
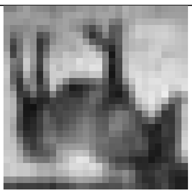
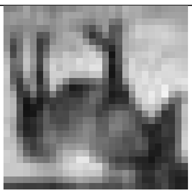
Input Image	**Truth Output**	**Model Output**
		
		
		
		
		
		

Table 2: Comparison between the true output and the neural network's output.

Information about the training process can be extracted from the Tensorboard logs shown below:

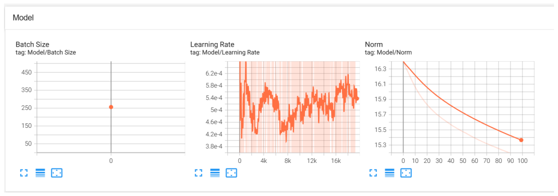


Figure 2: Model information

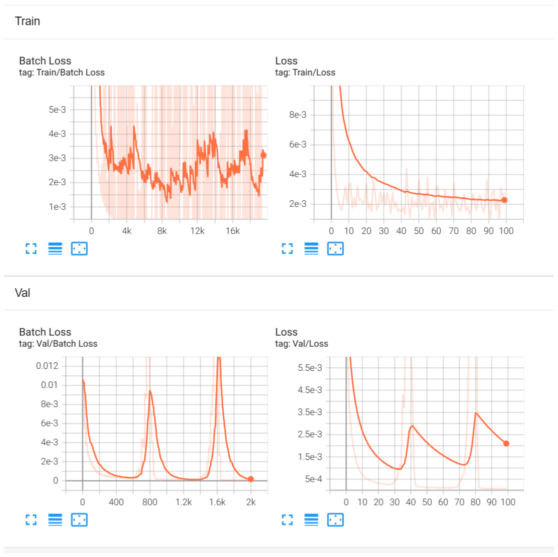


Figure 3: Training performance

4 Conclusion

In conclusion, the neural network implemented for this assignment has demonstrated excellent performance, successfully meeting the required objectives and producing the desired results. Its simple architecture and efficient training process have ensured that it stands up to the expectations set in the assignment guidelines.