Model Training and Inference Optimization

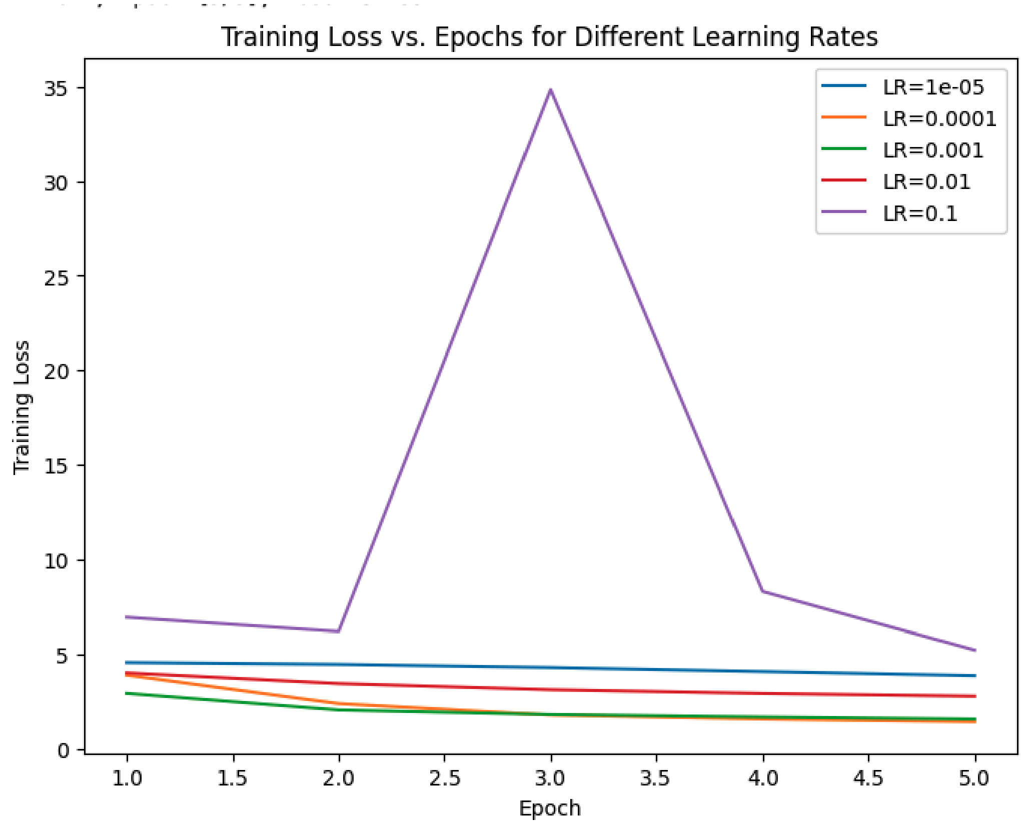
Objective: To train and optimize a model for inference on the CIFAR-100 dataset, focusing on optimizing for performance and measuring the inference time post-optimization.

Dataset and Model Selection

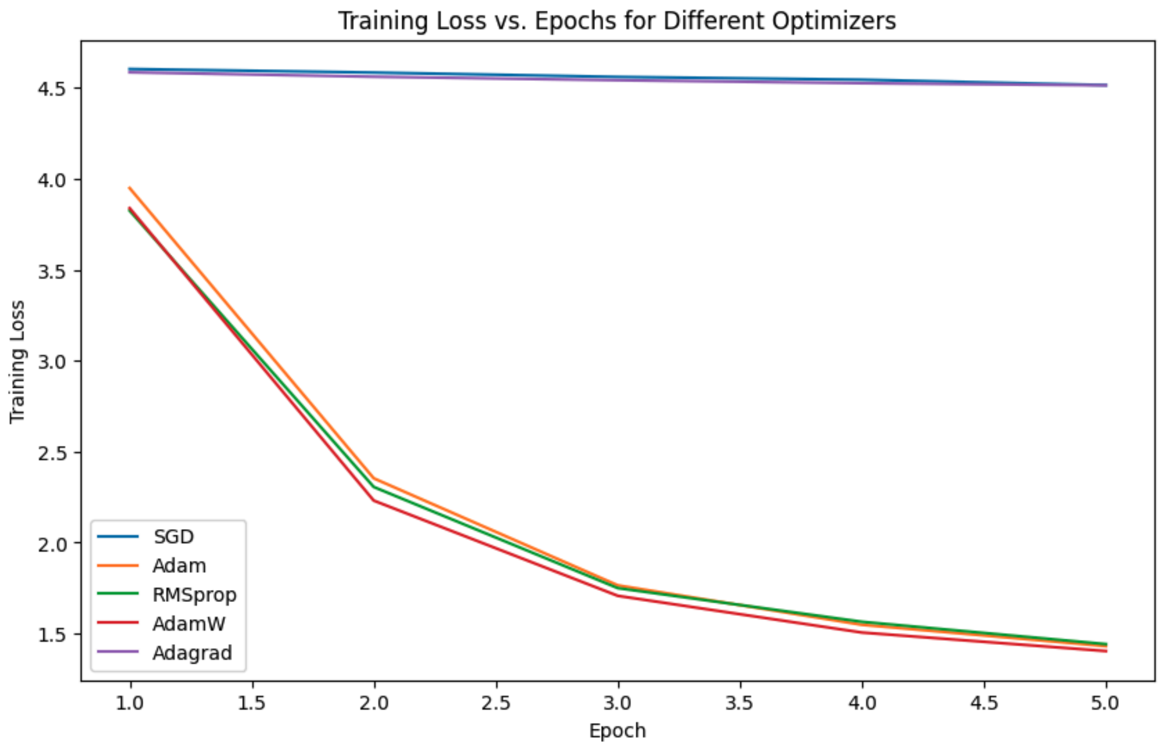
Dataset: The CIFAR-100 dataset was selected for this project due to its diversity and complexity, featuring 100 different classes and small 32x32 images, making it a good benchmark for classification tasks.
Transformation: Images were resized to 224x224 to match the input requirements of the chosen model.
Model: We chose efficientvit_m5.r224_in1k, a model from the TIMM library that balances efficiency with accuracy, particularly well-suited for image classification tasks. Also this model is light weight and ideal for learning purpose

Training Configuration

Learning Rate: After experimenting with different learning rates, we set the learning rate to a value that provided stability during training while allowing for efficient convergence.
The model was trained on different learning rates for 5 epochs and with batch size of 32.
LR= 0.0001 gave the best result after 5 epochs



Optimizer: Various optimizers were evaluated, including Adam, AdamW, SGD, and others. Based on performance and training stability, AdamW was selected as it gave good results in early iterations



Epochs: The model was trained for 50 epochs to ensure the network had ample time to converge, balancing computation cost with performance.

Inference Optimization Techniques

After training the model, we implemented multiple inference optimization techniques to reduce latency during inference:

TorchScript: We converted the trained model into a TorchScript format using both scripting and tracing methods. This conversion allowed the model to be saved and deployed independently from Python, significantly enhancing inference speed.

Dynamic Quantization: For CPU-based inference, dynamic quantization was applied to compress the model by reducing the precision of weights (to int8 in this case). This method effectively improved inference speed without a significant loss in accuracy.

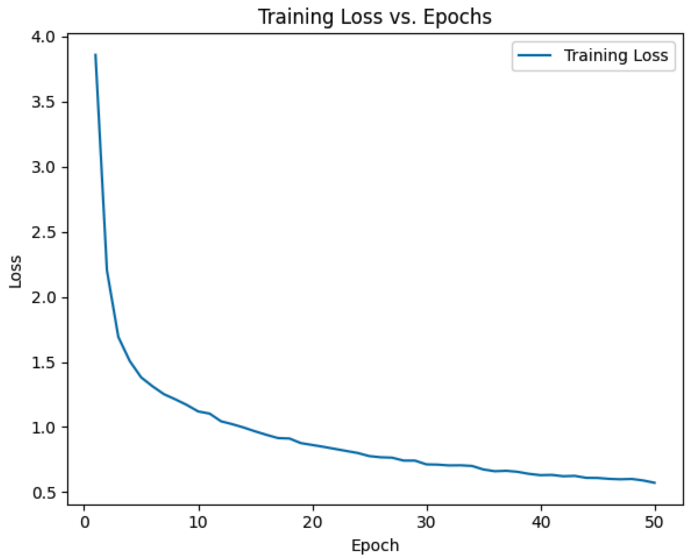
ONNX Export and Runtime: The model was exported to ONNX format and run using ONNX Runtime, a cross-platform, high-performance scoring engine for Open Neural Network Exchange (ONNX) models. This conversion facilitated compatibility across different environments and further enhanced performance.

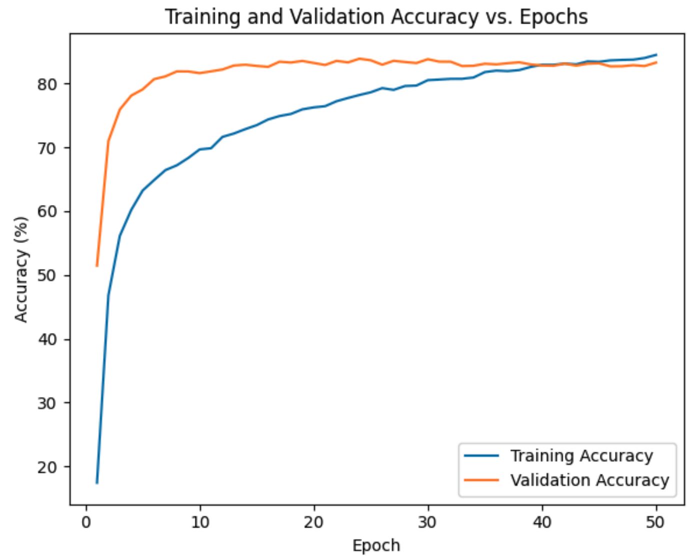
Mixed Precision (CUDA): For GPU-based inference, mixed precision with torch.cuda.amp was used, enabling the model to utilize both float16 and float32 operations, reducing memory usage and increasing speed.

Results and Inference Time

The final average inference times for each optimization technique were as follows:
After Training the model over 50 epochs, following were the results

```
Epoch [47/50], Loss: 0.5988, Accuracy: 83.71%
Validation Accuracy after Epoch [47/50]: 82.72%
Epoch [48/50], Loss: 0.6013, Accuracy: 83.75%
Validation Accuracy after Epoch [48/50]: 82.88%
Epoch [49/50], Loss: 0.5897, Accuracy: 84.01%
Validation Accuracy after Epoch [49/50]: 82.75%
Epoch [50/50], Loss: 0.5719, Accuracy: 84.50%
Model saved at epoch 50
Validation Accuracy after Epoch [50/50]: 83.29%
```





Training Loss: 0.5719
Average Epoch Accuracy: 84.50%
Validation Accuracy: 83.29%

Inference Time Summary:
TorchScript Inference Time: 0.0123 seconds
ONNX Inference Time: 0.0097 seconds
Mixed Precision Inference Time: 0.0299 seconds

Conclusion

By experimenting with different optimizers, learning rates, and inference optimizations, this assignement successfully developed an efficient classification model on CIFAR-100 with minimized inference latency. This demonstrates the value of model optimization techniques in improving deployment performance, particularly for real-world applications that demand rapid response time
Github Link: https://github.com/birdhunter22/cmpe_258_hw_1