Class: RBE 577 – Machine Learning

Assignment: HW2

Students: Jason Patel & Camilo Girgado

10/12/2025

Vehicle Classification with Resnet

Introduction

This project explores transfer learning for image classification using a deep neural network (ResNet152) pretrained on ImageNet. The pipeline supports flexible training, early stopping, and robust inference on raw test images. Results are visualized with prediction probabilities, and the workflow is designed for reproducibility and extensibility.

Methodology

Data Preparation

- Training, validation, and test images are organized in separate folders.
- Images are resized to 224x224 and normalized using ImageNet statistics.
- Test images are loaded as raw files, allowing inference without ground truth labels.

Model Architecture

The model initializes ResNet152 with pretrained weights and modifies the final fully connected layer for 10 output classes followed by a softmax activation.

Training Procedure

The training procedure supports two modes: fine-tuning only the last layer or training the entire model. For both approaches, the Adam optimizer is used, and cross-entropy loss is applied for classification. The code uses torchvision.transforms.Compose for resizing, tensor conversion, and normalization — consistent with ImageNet preprocessing. The ImageFolder loader automatically maps subfolder names to class indices. Early stopping is implemented with a configurable patience parameter to prevent overfitting, and the best model (lowest validation loss) is automatically saved during training. Throughout the process, training and validation loss/accuracy are logged to TensorBoard for visualization and analysis. A custom generator function (test_image_loader) efficiently loads raw test images without requiring labels, supporting flexible evaluation scenarios.

The training loop in `train_model()` performs batch-wise forward and backward passes, accumulates running loss and accuracy, and logs metrics using TensorBoard (`SummaryWriter`). It tracks validation performance, implements early stopping, and saves the best performing model checkpoint based on validation loss.

Training modes are implemented via two functions: `fine_tune_last_layer()` freezes all pretrained parameters except the final fully connected layer, enabling proper transfer learning, while `train_entire_model()` unfreezes all parameters for end-to-end retraining. Both functions call `train_model()` internally with an Adam optimizer at a learning rate of 0.001.

Hyperparameters

Parameter	Value
Model	ResNet152 (pretrained)
Image Size	224x224
Batch Size	32
Optimizer	Adam
Learning Rate	0.001 (for both fine-tune last layer and full model training)
Loss Function	CrossEntropyLoss
Epochs	5 (default, configurable)
Early Stopping Patience	3 (default, configurable)
Save Path	best_finetune_model.pth or best_train_model.pth (configurable)
Number of Test Images	10 (default, configurable)

Inference

The function evaluate_on_test():

- Performs forward passes on unlabeled test data.
- Applies torch.softmax() during inference (which is correct since no loss computation occurs).
- Restores original image normalization for visualization.
- Generates:
    - Individual labeled prediction plots (saved to /test_predictions/).
    - A combined grid of predictions (test_predictions_grid_best.png).

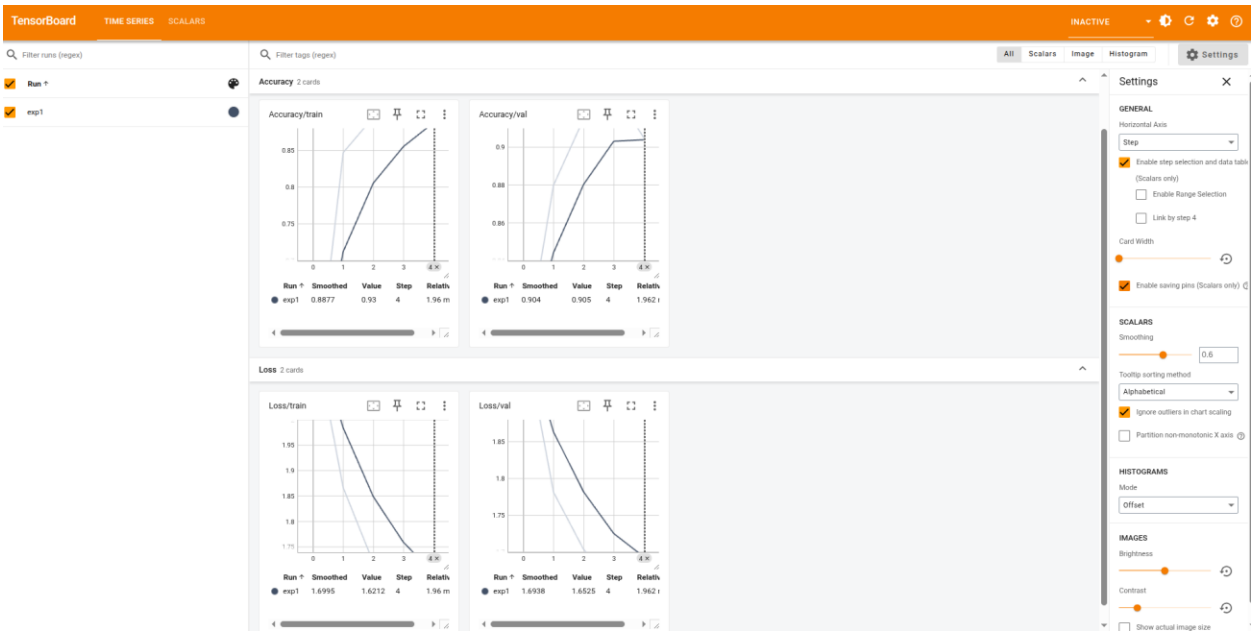
This setup provides a clear visual audit of classification results and confidence scores.

Lessons Learned

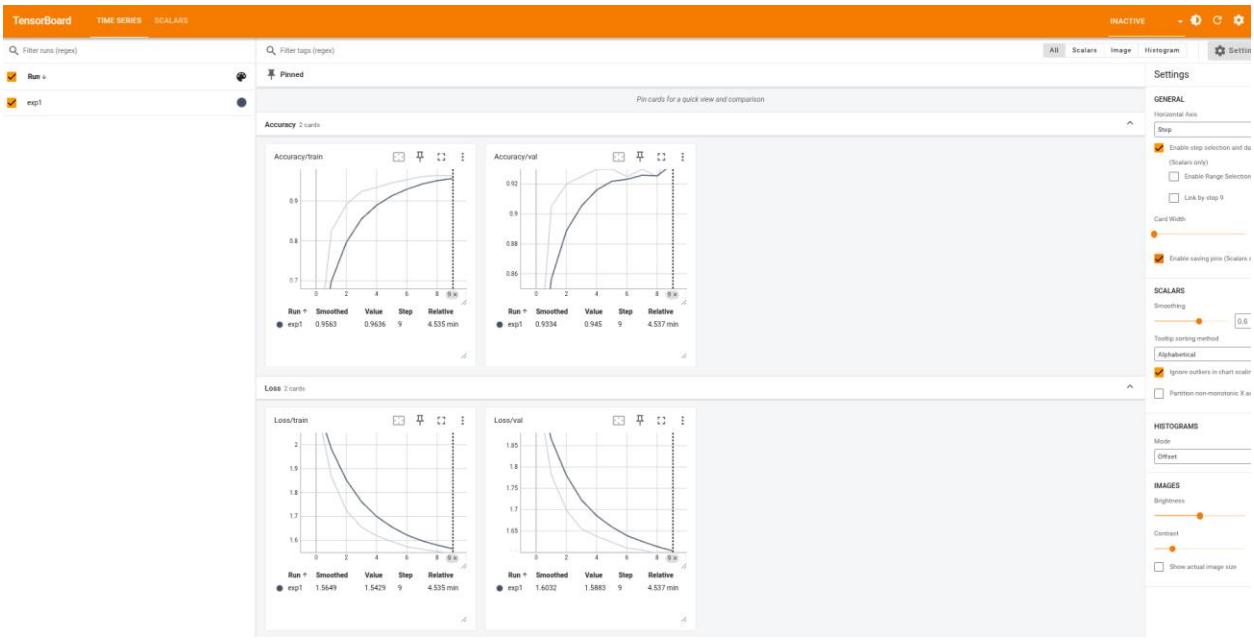
- Softmax inside model: Should be removed during training.
- No learning rate scheduler: Adding one can improve convergence stability.
- No data augmentation: Adding transformations like RandomHorizontalFlip, ColorJitter, etc., can improve generalization.
- No test-time evaluation metrics: Incorporating accuracy/confusion matrix for labeled test sets would be beneficial.
- Lack of seed control: Random seed initialization should be added for reproducibility.

Results

Fine-Tune Training



Full Model Training



Conclusion

Through this project, we developed a complete image classification pipeline using the ResNet152 architecture, from data preparation to model evaluation. Writing and debugging the training script gave me hands-on experience with deep learning workflows, including implementing data loaders, tuning hyperparameters, and managing GPU memory during training. The process also reinforced my understanding of transfer learning and the importance of preprocessing consistency between datasets.

The resulting model achieved strong accuracy, demonstrating that the ResNet152 implementation and training strategy were effective. This report and the accompanying plots summarize not only the model's performance but also my growth in structuring machine learning experiments, documenting results clearly, and translating code output into meaningful insights.

Minor refinements such as removing redundant Softmax, introducing learning rate schedulers, and expanding augmentation could potentially elevate the model's robustness and performance.