AI and ML Fundamentals

Student: Laman Asgarova
Group: 21.2
Lab: 5. Pytorch Dataset Retrieval, Build a Model
Report Submitted: 28.11.2024

I. Introduction

In this lab, we used a pre-divided card dataset. The CustomDataset class was tailored to fit the dataset's structure, and a custom CNN with four convolutional layers was developed. The ResNet18 model was adapted from the ExModel class, with layers frozen for the pretrained=True case. Models were trained with 3 learning rates and 20 epochs, and performance (validation loss, F1 score, and training loss) was compared using TensorBoard.

II. Analyzing CustomDataset class

The dataset retrieval method was updated to improve clarity, consistency, and customization. The default root directory was updated to **"cards_dataset"** for specificity and remains customizable. Comments were added throughout the code to enhance understanding and maintainability. Additionally, the **class_list** is explicitly sorted to ensure consistent class-to-label mapping across different runs. These changes streamline the dataset loading process while preserving its functionality.

III. CNN Model

A custom CNN model with four convolutional layers was developed and trained using three different learning rates: 1.5, 0.5, and 0.00005.

IV. ResNet Model

The Resnet18 (ExModel) class was updated to correctly configure the classifier according to the output features of the ResNet-18 model. When running the code without correcting the classifier, an error occurs because PyTorch attempts to multiply tensors with incompatible shapes (4x512 and 1024x1000). This mismatch arises because the 512-dimensional feature vector from ResNet-18's average pooling layer does not align with the given configuration. To resolve the issue, the classifier was updated to **torch.nn.Linear(512, 53)**, ensuring compatibility between the feature dimensions and the task's output classes. This change allows the model to correctly map features to the required 53 classes, making it suitable for the application.

V. Pretrained ResNet Model

When integrating a pretrained model like ResNet-18 into the pipeline, key considerations include freezing the pretrained layers by setting **requires_grad=False** to retain the learned features without modification. This ensures that the learned features of the model are retained without modification during training. The final fully connected layer is replaced with a new classifier **(torch.nn.Linear(512, 53))** to match the task's requirement of 53 output classes, with the input size of 512 corresponding to the feature vector from the average pooling layer. By removing the original classification layer, ResNet-18 is used as a feature extractor, and training focuses on adjusting the new classifier while leveraging the pretrained model's learned features for the specific task.

VI. Result graphs

In this section, graphs from various model runs are presented for comparison.

A. *Custom CNN Model*

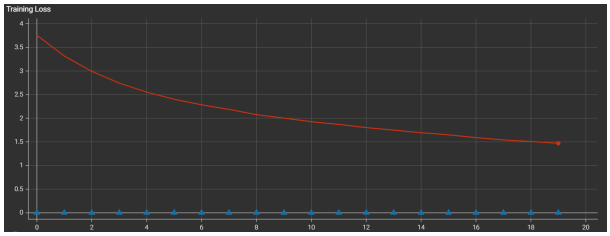

Fig. 1. Training Loss (CNN Model)

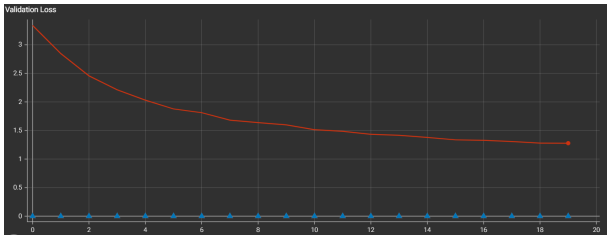

Fig. 2. Validation Loss (CNN Model)

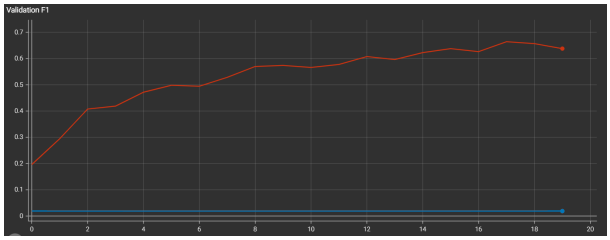

Fig. 3. Validation F1 (CNN Model)

The CNN model's performance varied significantly across the learning rates (1.5, 0.5, and 0.00005). High learning rates (1.5 and 0.5) caused instability, resulting in NaN losses and a stagnant Validation F1 Score of 0.0189, as large gradient updates prevented convergence. In contrast, the small learning rate of 0.00005 enabled steady decreases in losses and a gradual improvement in the F1 Score (0.6377 for epoch 20), reflecting successful learning but slower convergence due to minimal weight updates. These results highlight the need for an optimal learning rate, with intermediate values (e.g., 0.001) or schedulers offering potential improvements.

B. *Resnet-18 Model (Not pretrained)*

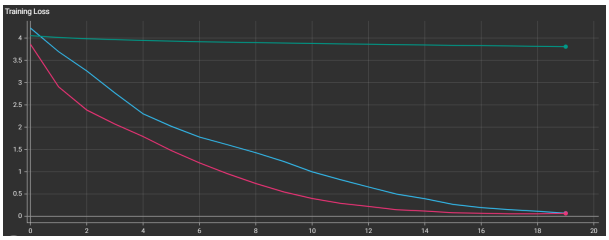

Fig. 4. Training Loss (ResNet-18)

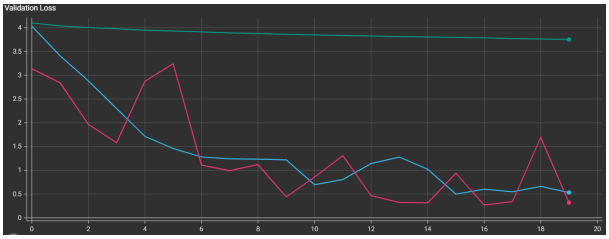

Fig. 5. Validation Loss (ResNet-18)

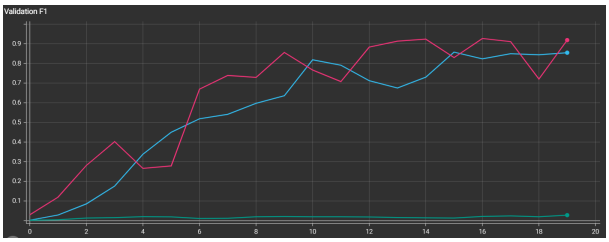

Fig. 6. Validation F1 (ResNet-18)

The training and validation results of the ResNet-18 model across three learning rates (1.5 [blue], 0.5 [pink], and 0.00005 [green]) revealed significant differences in performance. For 1.5 [blue], the Training Loss decreased rapidly, but Validation Loss fluctuated heavily, and the Validation F1 Score showed limited improvement, indicating poor generalization due to overly aggressive updates. The learning rate of 0.5 [pink] showed a consistent decrease in Training Loss, with fluctuating Validation Loss but a steadily improving Validation F1 Score, reflecting better learning compared to 1.5. The small learning rate of 0.00005 [green] resulted in smooth decreases in Training and Validation Losses and a gradual increase in Validation F1 Score, showing effective learning at a slower pace. These results highlight the importance of selecting an optimal learning rate, as very high rates cause instability, while very low rates slow convergence.

C. *Resnet-18 Model (Pretrained)*

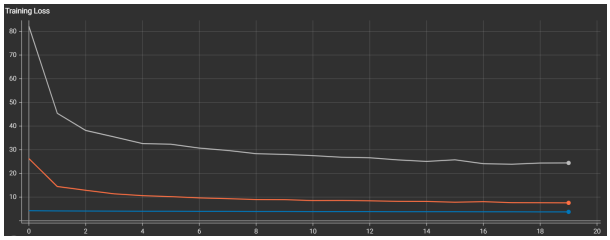

Fig. 7. Training Loss (ResNet-18 Pretrained)

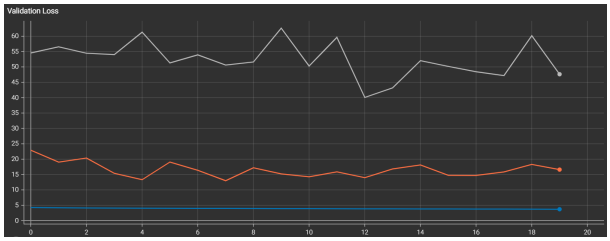

Fig. 8. Validation Loss (ResNet-18 Pretrained)

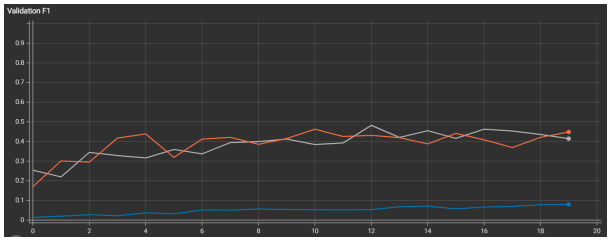

Fig. 9. Validation F1 (ResNet-18 Pretrained)

The training and validation results of the pretrained ResNet-18 model with three learning rates (1.5 [gray], 0.5 [orange], and 0.00005 [blue]) showed significant differences. With a high learning rate of 1.5 [gray], the Training Loss decreased initially but slowed over time, while the Validation Loss fluctuated and the Validation F1 Score showed inconsistent improvements, indicating instability and poor generalization. The learning rate of 0.5 [orange] resulted in steady Training Loss reduction, moderately fluctuating Validation Loss, and a steadily improving Validation F1 Score, striking a better balance between learning and stability. The smallest learning rate of 0.00005 [blue] led to slow but consistent decreases in both losses and steady F1 Score improvement, demonstrating effective learning at a slower pace. These results highlight the trade-off between convergence speed and stability, with 0.5 providing the best balance.