Muhammad Nur Maajid_1103228145

Module 7 delves into PyTorch Experiment Tracking. Throughout the development of the FoodVision Mini project (an image classification model to classify images of pizza, steak, or sushi), numerous models have been trained. Until now, tracking has been managed through Python dictionaries or metric comparisons during training. However, for operating dozens of different models simultaneously, a more systematic approach, such as experiment tracking, is essential.

The Significance of Experiment Tracking in Machine Learning:

- Experimentation is crucial in machine learning. To discern what works and what doesn't, different experiment outcomes need to be methodically tracked.

1. What is Experiment Tracking:

- Machine learning and deep learning are fundamentally experimental.
- Experiment tracking is necessary to create various models and monitor the outcomes of different data combinations, model architectures, and training methods.
- It aids in identifying successful and unsuccessful approaches.

2. Why Track Experiments:

- For a few models, results can be tracked via printouts and dictionaries.
- However, as the number of experiments grows, this approach can become impractical.
- Experiment tracking becomes increasingly important as managing diverse experiments can get complicated.

3. Various Methods for Tracking Machine Learning Experiments:

- There are several methods for tracking machine learning experiments, including Python dictionaries, CSV files, printouts, TensorBoard, Weights & Biases Experiment Tracking, and MLFlow.
- Each method has its strengths and weaknesses, as well as different costs.

0. Preparation:

1. Download necessary modules by checking and installing appropriate versions of PyTorch and torchvision.
2. Prepare code for device-agnostic usage by determining the device (CPU or GPU).
3. Create a set_seeds() function to set random seeds for torch operations.

1. Acquiring Data:

1. Download the "pizza_steak_sushi" dataset used for the FoodVision Mini experiments.

2. Create a download_data() function to download and extract the dataset if not already present.
3. Configure the dataset path.

2. Creating Datasets and DataLoaders:

1. Configure transformations to convert images into tensors and normalize them according to the ImageNet format.
2. Create DataLoaders using manually created transformations.
3. Create DataLoaders using automatic transformations from pretrained weights.

2.1. Manual Creation of DataLoaders:

1. Configure directory paths for training and testing data.
2. Implement ImageNet normalization as part of the manual transformations.
3. Build a manual transformation pipeline using torchvision.transforms.Compose().
4. Create DataLoaders using the create_dataloaders() function from the data_setup module.

2.2. Automatic Creation of DataLoaders:

1. Configure directory paths for training and testing data.
2. Select pretrained weights (e.g., EfficientNet_B0) from torchvision.models.
3. Obtain automatic transformations by calling the transforms() method of the selected weights.
4. Create DataLoaders using the create_dataloaders() function from the data_setup module.

3. Obtaining a Pretrained Model:

1. Download pretrained weights for the EfficientNet_B0 model from torchvision.
2. Construct the model with these weights, then freeze the base layers and modify the classifier head to suit the number of classes.

4. Training the Model and Tracking Results:

1. Specify loss function (CrossEntropyLoss) and optimizer (Adam) for training the model.
2. Set up a SummaryWriter for tracking experiment results using TensorBoard.
3. Train the model using the train() function and track results with SummaryWriter.

5. Monitoring Model Results on TensorBoard:

1. Use the SummaryWriter() class to store model results in the TensorBoard format by default in the "runs/" directory.
2. TensorBoard is a visualization tool created by the TensorFlow team for viewing and inspecting information about models and data.
3. It allows for interactive visualization of results during and after training.
4. Accessing TensorBoard:

    - In VS Code: Press SHIFT + CMD + P, search for the command "Python: Launch TensorBoard."
    - In Jupyter and Colab Notebooks: Ensure TensorBoard is installed, load it with %load_ext tensorboard, and view results with %tensorboard --logdir DIR_WITH_LOGS.
    - Experiments can also be uploaded to tensorboard.dev for public sharing.

6. Creating a Helper Function to Build SummaryWriter Instances:

1. The SummaryWriter() class records information to a directory specified by the log_dir parameter.
2. Create a helper function, create_writer(), to make a SummaryWriter() instance with a customized log_dir for each experiment.
3. The log_dir includes a timestamp, experiment name, model name, and additional (optional) information.
4. Customize the log directory to track various details like experiment date, experiment name, model name, and additional information.
5. Update the training function (train()) to accept a writer parameter so each experiment can use a different SummaryWriter() instance.

6.1. Updating the train() Function to Include a Writer Parameter:

1. Add a writer parameter to the train() function, enabling the use of different SummaryWriter() instances for each experiment.
2. When calling train(), different writers can be passed for each experiment, resulting in different log directories.
3. The train() function can now actively update the SummaryWriter() instance used each time it's called.

7. Setting Up a Series of Modeling Experiments: In this step, a series of modeling experiments are conducted to improve the FoodVision Mini model without making it overly large. This approach involves trying various combinations of data, models, and epochs. Key steps include:

7.1. Modeling Experiments:

- The big question in machine learning is what type of experiments to run.
- There are no limits to the experiments that can be conducted, making machine learning both exciting and daunting.
- In this step, don a scientist's coat and follow the machine learning practitioner's motto: experiment, experiment, experiment!
- Every hyperparameter becomes a starting point for different experiments. Possible experiments to run include:
    - Changing the number of epochs.
    - Altering the number of hidden layers/units.
    - Varying the amount of data.
    - Modifying the learning rate.

    - Trying different data augmentations.
    - Selecting various model architectures. With practice and running many different experiments, intuition about what might help the model will start to develop.

7.2. Experiments to be Conducted:

- The goal is to improve the FoodVision Mini model without making it too large.
- The ideal model achieves high test set accuracy (90%+) but doesn't take too long to train/infer.
- Experiments include combinations of:
    - Different amounts of data (10% vs. 20% of Pizza, Steak, Sushi).
    - Different models (EfficientNetB0 vs. EfficientNetB2).
    - Varying training durations (5 epochs vs. 10 epochs).
- Experiments are conducted progressively, increasing the amount of data, model size, and training duration.

7.3. Downloading Different Datasets:

- Before running a series of experiments, ensure datasets are ready.
- Two forms of training sets are needed:
    - A training set with 10% of Pizza, Steak, Sushi images from Food101 (already created).
    - A training set with 20% of Pizza, Steak, Sushi images from Food101.
- Both datasets are downloaded from GitHub.

7.4. Transforming Datasets and Creating DataLoaders:

- Create transformations to prepare images for the model.
- Create DataLoaders with a batch size of 32 for all experiments.

7.5. Creating Feature Extracting Models:

- Create two feature-extracting models: EfficientNetB0 and EfficientNetB2.
- Freeze base layers (features) and modify the classifier head to suit the image classification problem.
- EfficientNetB2 requires adjustments to the number of final input features.

7.6. Setting Up Experiments and Training Code:

- Create two lists and one dictionary for the epochs to be tested, models to be tested, and different DataLoaders.
- Run experiments by changing the model, number of epochs, and DataLoaders.
- Save trained models after each experiment for later use in predictions.

8. Reviewing Experiments on TensorBoard:

- Use TensorBoard to visualize experiment results.
- Observe that the EffNetB0 model trained for 10 epochs with 20% data achieves the lowest test loss.
- Visualize experiment results on tensorboard.dev for public sharing.

9. Loading the Best Model and Making Predictions:

- Analyze TensorBoard logs and find that the eighth experiment achieved the best overall results (highest test accuracy, second-lowest test loss).
- The best model is EffNetB2 with fine-tuned parameters, 20% training data of pizza, steak, sushi, trained for 10 epochs.
- Although the results are not much better than other models, the same model on the same data achieved similar results in half the training time.
- Load the best saved model and check the model file size.
- Make predictions on test images never seen before.

9.1. Predicting on Custom Images with the Best Model:

- Display predictions on a custom image ("pizza dad") using the best model.
- The best model successfully predicts "pizza" with high confidence (0.978)