109550198,卜銳凱

INTRODUCTION TO MACHINE LEARNING
FINAL PROJECT

- Environment details (5%)
  - **Python version**

  - **Framework**

    The code I've provided is written in Python and uses the PyTorch framework. PyTorch is an open-source machine learning library developed by Facebook's AI Research lab. It's widely used for applications such as computer vision and natural language processing.

    Key aspects of PyTorch visible in your code include:

    PyTorch Model Definition: The use of models.resnet152(pretrained=True) indicates the use of PyTorch's torchvision module, which provides pre-trained models for computer vision tasks.

    Tensors and Operations: Operations like .to(device), torch.max(), and torch.no_grad() are specific to PyTorch and are used for tensor computations and model inference.

    Data Handling and Transforms: The use of transforms.Compose, transforms.Resize, transforms.ToTensor, and transforms.Normalize are part of PyTorch's torchvision.transforms module, used for preprocessing and transforming image data.

    DataLoader: The DataLoader class is a PyTorch utility that provides an iterable over a dataset, with support for batching, sampling, shuffling, and multiprocess data loading.

    Training and Optimizers: Although not explicitly shown in the snippet you provided for inference, references to training elements like optim.Adam for the

optimizer and nn.CrossEntropyLoss for the loss function are indicative of PyTorch's use for model training.

- **Hardware**

  The code I've provided is designed to run on a variety of hardware configurations, but it specifically checks for and utilizes a CUDA-capable GPU if available:

  device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

  This line in the code automatically sets the device to a GPU with CUDA support (if one is available) or falls back to the CPU if a GPU isn't available or doesn't have CUDA support.

- Implementation details (15%)
  - **Model architecture**

    The model architecture I am using is ResNet152. This is one of the deeper models available in the ResNet (Residual Network) family, which is known for its ability to train very deep neural networks by utilizing skip connections, or shortcuts to jump over some layers.

    ResNet152 is implemented in PyTorch and can be accessed via the torchvision.models module, which provides a set of pre-defined architectures with the option for pre-trained weights. The model is commonly used for tasks that require the extraction of intricate patterns and details from images, such as fine-grained image classification, object detection, and more.

  - **Hyperparameters**

    In the context of the code i've provided earlier, the hyperparameters refer to the settings that can be adjusted to control the learning process of the neural network model. Based on the code snippets, here are the hyperparameters used:

    Learning Rate: The learning rate for the Adam optimizer is set to 5.5e-5. This is a crucial hyperparameter that determines the step size during the optimization process.

    Batch Size: I am using a batch size of 32 for training. Batch size affects memory usage and how accurately the gradient approximates the true gradient of the entire dataset.

    Number of Epochs: The number of epochs is set to 20. An epoch is one complete pass through the entire training dataset.

Optimizer: The choice of optimizer itself is a hyperparameter. i have selected the Adam optimizer, which is known for combining the best properties of the AdaGrad and RMSProp algorithms.

Dropout Rate: In the final fully connected layer, a dropout rate of 0.5 is used to prevent overfitting.

Weight Initialization: While not explicitly a hyperparameter i set, using a pre-trained model involves the hyperparameter of initial weight values, which have been learned from the ImageNet dataset.

Image Size: You are resizing images to 128x128. This determines the input size for the network.

Data Augmentation Parameters: For the training dataset, random rotations of 15 degrees and random affine transformations with translations up to 0.1 and scale changes between 0.8 and 1.2 are used as a form of data augmentation.

Loss Function: The loss function used is CrossEntropyLoss, which is standard for multi-class classification problems.

These hyperparameters can significantly influence the performance of my model

- **Training strategy**

Transfer Learning: The model being used is ResNet152, which is pre-trained on the ImageNet dataset. This is a form of transfer learning where the model has already learned rich feature representations from a large and diverse set of images, which can be leveraged to improve performance on a new task with potentially less data.

Fine-Tuning: In the provided code, the last fully connected layer of the pre-trained ResNet152 model is replaced to match the number of classes in the new dataset. The entire model is then fine-tuned on the new dataset, which allows both the newly initialized layers and the pre-trained layers to adjust to the new data.

Data Augmentation: The transformations applied to the training data include random rotations and affine transformations. These augmentations help prevent overfitting by introducing a variety of transformations that the model needs to become invariant to, thereby improving its generalization ability.

Normalization: The input data is normalized using mean and standard deviation values of [0.5, 0.5, 0.5] for all three color channels (RGB). This normalization helps in accelerating the convergence of the training process.

Loss Function: Cross-Entropy Loss is used, which is standard for classification tasks. It measures the performance of a classification model whose output is a probability value between 0 and 1.

Optimizer: The Adam optimizer is used, which is an adaptive learning rate optimization algorithm designed specifically for training deep neural networks. It combines the advantages of two other extensions of stochastic gradient descent, namely AdaGrad and RMSProp.

Model Saving Strategy: The code includes a strategy to save the model weights whenever there is an improvement in accuracy. This allows for the retention of the best model throughout the training process.

GPU Utilization: Training is performed on a GPU if available, which significantly speeds up the training process due to the parallel processing capabilities of GPUs.Evaluation During Training: Although not shown in the snippets, typically, evaluation on a validation set during training would be part of the strategy to monitor for overfitting and to make decisions about early stopping or learning rate adjustments.

Regularization: Dropout is used in the final layer with a rate of 0.5 to reduce overfitting by preventing complex co-adaptations on training data.

It's important to note that this strategy assumes that there's a separate test set or another form of validation like cross-validation, which was not explicitly mentioned in the code snippets. Without this, it would be challenging to gauge the true performance and generalization ability of the model.

- Experimental results (15%)
  - **Evaluation metrics**

    Accuracy: The most straightforward metric, it measures the ratio of correctly predicted observations to the total observations. It's useful when the target classes are well balanced.

    Precision: Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. It is a measure of a classifier's exactness. High precision indicates a low rate of false positives.

  - **Learning curve**

I Detect Overfitting: the training performance continues to improve, but the validation performance gets worse, the model is starting to memorize the training data and is performing poorly on the validation data. So I only use the training data.

- **Ablation Study**

  For an ablation study:

  Train and evaluate my model with the scheduler and optim.SGD.

  Train and evaluate my model without the scheduler (i.e., with a constant learning rate).

  Train and evaluate my model by replacing optim.SGD with another optimizer.

  By comparing the performance of your model under these different conditions, I better understand the impact of the learning rate scheduler and the choice of optimizer on my model's accuracy.

- Bonus (5%)
  - Such as comparisons, clear plots, methods/papers review and discussion, etc.

PS:

Path

Inside /content/data/data I create a folder which is test1 , after I put test inside this folder.