Due Date : March 1st, 2024 at 11 :00 pm

Overview

In this assignment, you will perform Human Activity Recognition (HAR) using deep learning techniques. You will be working with time series data collected from smartphone accelerometers, which include x, y, and z axis acceleration data, timestamps, and person IDs. The primary goal is to develop models that can accurately classify six types of activities : Walking, Jogging, Sitting, Standing, Climbing Upstairs, and Going Downstairs. This assignment is structured in three parts :

1. Implementing and testing a Multilayer Perceptron (MLP) on the HAR time series data. 2. Implementing and testing a 1D Convolutional Neural Network (1D CNN) on the same dataset. 3. Advanced implementation and experimentation, where you will explore more complex models and techniques to enhance performance and interpretability.

This assignment is designed to test your skills in data preprocessing, model building, hyperparameter tuning, and performance analysis. It will also provide you with practical experience in handling real-world time series data and insights into the challenges and considerations in building effective HAR systems.

Coding instructions You will be required to use PyTorch to complete all questions. Moreover, this assignment **requires running the models on GPU** (otherwise it will take an incredibly long time) ; if you don't have access to your own resources (e.g. your own machine, a cluster), we recommend using Google Colab. You can download a notebook for this assignment here. Use the notebook for handling the data and have an outline for coding.

Submission You are expected to submit `mlp.py` and `cnn.py` to gradescope for autograding, as well as a PDF report for experimental and open-ended problems.

PDF Report Instructions : The PDF report should document your experimental setup, results, and analysis for each part of the assignment. Include sections on model architecture, hyperparameter choices, training process, challenges encountered, solutions tried, and insights gained. Graphs and tables to support your analysis are highly encouraged. **Important note :** Do not forget assigning pages to the corresponding questions to avoid any loss of points.

Data Preparation The notebook will handle the following steps without requiring data wrangling : - Importing, converting, and reformatting accelerometer data into a time-sliced representation. - Visualizing the accelerometer data.

Overview of the Dataset The dataset includes : **Accelerometer Data** from a smartphone :

X, Y, Z axis acceleration, Timestamp, Person ID, **Activities** Six activities including Walking, Jogging, Sitting, Standing, Upstairs, and Downstairs, **Models** : The task is to train models to classify between the 6 activities using : 1. Multi-Layer Perceptron (MLP). 2. 1D Convolutional Neural Network (time-series CNN).

Question 1. (25 points) **(Implementing and Testing a MLP on HAR Time Series)**

(1.1) **MLP Model Implementation :** Implement a Multilayer Perceptron (MLP) for the HAR dataset. Follow the architecture outlined below for your implementation :

— Begin with a flatten layer (`nn.Flatten`) to reshape the input tensor into a 1D tensor, preparing it for processing by fully connected layers.

— Construct four fully connected layers (`nn.Linear`), each followed by a ReLU activation function (`F.relu`), except after the last layer. The input to the first layer should be `time_periods * 3` (accounting for the three acceleration axes over the specified time periods), with an output of 100 features. The second and third layers should each take 100 input features and also output 100 features. The final layer should transform these 100 input features into `n_classes` outputs, corresponding to the predicted probabilities for each activity class.

The `forward` method of your model should apply these layers in sequence to the input tensor `x`, ensuring the data is correctly reshaped or flattened as required for each layer's input.

Once your model is defined, create an instance of it by passing `TIME_PERIODS` and `n_classes` as arguments. Move the model to the specified `device`, (which is GPU since the training on CPU will take too long), based on your setup.

(1.2) **Hyperparameter Tuning :** Experiment with different architectures by varying the number of hidden layers and batch sizes. Explore the use of different activation functions (e.g., ReLU, sigmoid, tanh) and learning rates. Test at least three different configurations and discuss the impact of these hyperparameters on your model's performance.

(1.3) **Model Performance and Hyperparameter Analysis :** Analyze the model's performance across different hyperparameters. Generate a confusion matrix for the best-performing configuration to evaluate accuracy, loss convergence, and activity-specific performance.

(1.4) **Confusion Matrix Evaluation :** Evaluate the importance of the confusion matrix in understanding your model's strengths and weaknesses in classifying different activities.

Question 2 (25 points)**. (Implementing and Testing a 1D CNN on HAR Time Series)**

(2.1) **1D CNN Model Implementation :** Implement a 1D Convolutional Neural Network using PyTorch, tailored specifically for the HAR dataset. Your implementation should include the following architecture :

— Four 1D convolutional layers (`nn.Conv1d`), with ReLU activation functions (`F.relu`) after each convolutional layer. The first convolutional layer should accept `n_sensors` input channels and output 100 channels. The second layer should take these 100 input channels and also output 100 channels, while the third and fourth layers should increase the output channels from 100 to 160. Set the kernel size for all convolutional layers to 10.

- After the second convolutional layer, include a max pooling layer (`nn.MaxPool1d`) with a pool size of 3 to reduce the dimensionality of the data.
- Following the last convolutional layer, apply an adaptive average pooling layer

  (`nn.AdaptiveAvgPool1d`) that reduces the output to a size of 1 along the time dimension, effectively summarizing the features extracted by the convolutional layers.
- Introduce a dropout layer (`nn.Dropout`) with a dropout rate of 0.5 after the adaptive average pooling layer to prevent overfitting by randomly omitting a subset of features during training.
- Finally, add a fully connected layer (`nn.Linear`) that takes the flattened output from the dropout layer and produces `n_classes` output values, corresponding to the predicted probabilities for each activity class.

In the `forward` method of your model, ensure that these layers are applied in sequence to the input tensor `x`, with appropriate reshaping or flattening as needed to match the input dimensions of each layer. Document the implementation details and any challenges encountered in integrating these specific layers and configurations in your model.

(2.2) **Hyperparameter Tuning :** Adjust the architecture by changing the number of convolutional layers, the size of convolutional filters, and the stride. Experiment with different activation functions (e.g., ReLU, LeakyReLU) and pool types (max pooling, average pooling). Evaluate at least three configurations to understand their effects on model performance.

(2.3) **Confusion Matrix Evaluation :** Use the confusion matrix to detail your model's performance on each activity type, emphasizing the advantages of CNNs in capturing time-dependent features. Discuss the optimal configuration and why certain hyperparameters were more effective, supported by the confusion matrix and performance metrics.

Question 3 (50 points)**. (Reflections of the Implementation and Experimentation)** This part of the assignment is designed to experiment more on the model development and evaluation. The tasks are divided into several key areas :

(3.1) **Data Augmentation :** Implement data augmentation techniques to enhance the robustness and generalization of your models. Use the following functions as a basis for your implementation, adjusting parameters as necessary for the HAR dataset :

(a) Noise Addition : Add Gaussian noise to the data to mimic real-world variability and sensor imperfections.
(b) Time Shifting : Shift data points in time to simulate the effect of time lags.
(c) Discuss augmentation techniques used in similar contexts, such as ImageNet, and their potential applicability to time-series data.
(d) Implement one other ad-hoc technique that can mitigate the challenges you've had with this specific dataset.

(3.2) **Baseline Comparison :** Considering the baseline is a very primary step in evaluating a classification technique.

(a) Argue what the random baseline should be for this task.
(b) Compare your models' results with a random baseline to establish their effectiveness.

(c) Train models for over 500 epochs to investigate potential overfitting and document your findings.

(3.3) **Batch Normalization and Layer Normalization :** For the CNN model, experiment with both batch normalization and layer normalization. Justify your choice of the layer(s) that you want to do the normalization on.

(3.4) **Experimentation with Optimizers :** Compare the performance of Adam and AdamW optimizers across different learning rates (0.1, 0.01,0.001) on your models. Discuss the impact of optimizer choice and weight decay on model performance.

(3.5) **Model Scaling :** Focus on scaling down both the CNN model to examine the impact of model complexity on performance. Implement the following specific modifications in PyTorch for your CNN model :

(a) Reduce the Number of Filters in Convolutional Layers : Scale down the convolutional layers by adjusting the number of filters. Change the first and second convolutional layers to have 50 filters each, and the third and fourth layers to have 80 filters each.

(b) Simplify or Remove Regularization : To prevent underfitting in the smaller model, adjust the dropout rate. Experiment with a dropout rate of 0.3 and entirely remove dropout to observe its effect.

(c) Reflect on these changes by monitoring training and validation performance, looking for signs of overfitting or underfitting, and noting the model's learning speed. Document the effects of downscaling on model accuracy and computational efficiency. Optionally, explore the removal of layers for further simplification and its impact.

(d) Data Scaling : There are different methods of downscaling data. One is downsampling by reducing the temporal resolution : Considering the dataset's 20Hz sampling rate, explore temporal downsampling options to reduce data volume without significantly impacting activity pattern recognition :

— Reduce the sampling rate by selecting every 2nd sample for a new rate of 10Hz, and every 5 sample dor a new rate of 4 Hz. Reflect on how this affects the dataset size and model accuracy.

(3.6) **Sampling Strategies :** Given the unbalanced nature of the HAR dataset, with some activities being more frequent than others, implement and compare two sampling strategies :

(a) Random Sampling : As a baseline, use random sampling to select data points for training. Analyze its impact on model performance across different activities.

(b) Oversampling Minor Classes : Specifically increase the representation of minor classes in the training data by sampling them more frequently than their natural occurrence rate. This can help in addressing class imbalance.

(c) Analyze the effectiveness of sampling from various densities and its impact on the model's ability to learn from unbalanced data.

Note : Document all experiments, observations, and conclusions in your PDF report. Include graphs, confusion matrices, and tables where applicable to support your analysis.