Assignment - Part (1)

Objective:

Design and implement a pipeline for radar point cloud data preprocessing and train a deep learning model to classify motion patterns into **normal activity** and **fall activity**. This assignment tests your skills in:

1. Point Cloud Data Preprocessing
2. Feature Engineering
3. Deep Learning Model Design and Training
4. Performance Evaluation and Visualization

Problem Statement:

You are given radar point cloud data in `.npy` format, where each file contains motion patterns captured at 10 frames per second. Your task is to:

1. Preprocess the radar data to extract meaningful features.
2. Apply transformations using a **rotation matrix** for tilt correction.
3. Design and train a deep learning model for binary classification (**normal vs. fall**).
4. Evaluate the model's performance on unseen data.
5. Visualize and interpret the results.

Assignment Tasks:

Task 1: Data Preprocessing

- Consider the data files present in the link

  https://drive.google.com/drive/folders/110VC1af8lXckWtDrb96uJzWRIeTIlL9W?usp=sharing

  - `normal.npy` (contains normal activities).
  - `falls.npy` (contains fall activities).

Step 1: Initialization

- **Parameters**: Define these key values in your implementation:
  - `frames_per_pattern = 10`: Each motion pattern consists of 10 consecutive radar frames (e.g., 10 fps for 1 second).

- `points_per_frame = 64`: Target number of points in each frame after oversampling.
- `features_per_point = 4`: Retain the first 4 features of each point (`delta_x`, `delta_y`, `delta_z`, Doppler).
- `tilt_angle = -10.0`: Radar tilt angle in degrees.
- `height = 1.8`: Radar height in meters.

- **Rotation Matrix**:
  Construct a rotation matrix to adjust coordinates based on the tilt angle

$$R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix}$$

where θ is the tilt angle in radians.

Step 2: Load Data

- Load the `.npy` radar point cloud file using `numpy.load()`.
- The data is structured as a list of frames, where each frame contains multiple radar points. Each file contains point cloud information in the format: `[frame_number, point_ID, target_ID, centroidX, centroidY, centroidZ, centroidVelocityX, centroidVelocityY, centroidVelocityZ, range, azimuth, elevation, Doppler, SNR, noise_level]`.
- Group the data into motion patterns of size `frames_per_pattern` (e.g., 10 consecutive frames per pattern).

Step 3: Feature Extraction and Transformation

For each motion pattern:

1. Extract the centroid information (`centroidX`, `centroidY`, `centroidZ`) from the first frame.
2. Apply the **rotation matrix** to the centroid coordinates:
   Add the radar height to `Z'` for adjusted elevation.
3. For each point in all frames:
   - Convert spherical coordinates (`range`, `azimuth`, `elevation`) to Cartesian coordinates (`x`, `y`, `z`).
   - Apply the **rotation matrix** to transform these Cartesian coordinates.
   - Compute the relative features with respect to the centroid:
     - `delta_x = pointX - centroidX`
     - `delta_y = pointY - centroidY`
     - `delta_z = pointZ`
     - `delta_D` (Doppler shift).
     - `pointRCS`: Compute using below formula

Point RCS=4 · 10 · log10(range)+SNR · 0.1+noise_level · 0.

4. Retain only the first 4 features (`delta_x`, `delta_y`, `delta_z`, Doppler) for each point.

Step 4: Oversampling

Oversample each radar frame to ensure exactly 64 points per frame:

a. Use **mean padding** to add points when a frame has fewer points.
b. Downsample randomly if a frame exceeds 64 points.

Task 2: Deep Learning Model

- Design a **3D Convolutional Neural Network (3D-CNN)** to classify motion patterns into **normal activity (0)** and **fall activity (1)**.
- Model Input Shape: (10,64,4)(10, 64, 4)(10,64,4), where:
  - 10: Number of frames per pattern.
  - 64: Number of points per frame.
  - 4: Features per point (ΔX,ΔY,Z,Doppler)
- Model Requirements:
  - Use 3D convolutional layers to extract spatiotemporal features.
  - Include batch normalization and dropout for regularization.
  - Use a fully connected layer with a **sigmoid** activation function for binary classification.

Task 3: Training, Evaluation and Visualization

- Split the preprocessed data into training, validation, and test sets (e.g., 70%-20%-10%).
- Train the model using:
  - **Binary Cross-Entropy Loss**.
  - **Adam Optimizer**.
- Evaluate the model on the test set using:
  - **Accuracy**.
  - **Precision, Recall, and F1-Score**.
  - **ROC-AUC** curve.
  - **Confusion Matrix**.
- Visualization: Provide the following visualizations:
  - **Training and Validation Curves**: Plot loss and accuracy over epochs.
  - **Point Cloud Visualization**: Show the motion patterns for a normal activity and a fall activity using scatter plots.

Submission Requirements:

1. Python scripts for:
    - Data preprocessing.
    - Model training and evaluation.
2. A report including:
    - Description of your approach.
    - Key observations and findings.
    - Plots and visualizations.
3. A trained model file (`.h5` or equivalent format).
4. Instructions to reproduce your results.

Assignment - Part (2)

Deployment of the above models, code and data.

Requirements:

1. Local Deployment of the model

2. The model should be deployed on a local container (using docker).

3. Data should be stored locally and the deployed model should be called from the container with a script to be run on the local data and give the output back to the container.

4. Documentation - on the model, the code, the deployment + docker.

Main Requirement:

Ideally, the reviewer should be able to pick up your zip file (submitted file for this part of the assignment) and be able to follow your documentation to deploy this container on their local machine with proper outputs as mentioned in the first part of the assignment

Bonus:

If you have any free tiers of cloud deployment opportunities (AWS, GCP, Azure) and you would like to deploy the container there and run it and share the links with us for review - that is indeed a bonus added capability which will help your candidacy.