In the Name of God

Simulation Report

Summary of the Paper:

In the paper *"Transfer Learning for UWB Error Correction and (N)LOS Classification in New Environments"*, two neural networks are introduced:

1. Deep Neural Network (DNN):
   - Input: 12 extracted features from the CIR (Channel Impulse Response) radio signal.
   - Task: Classification of environments into LoS (Line-of-Sight) or NLoS (Non-Line-of-Sight).
2. Convolutional Neural Network (CNN):
   - Input: Raw CIR signal.
   - Task: Automatic feature extraction and classification into LoS or NLoS.

A key contribution of this paper is the use of Transfer Learning, enabling pre-trained weights in one environment to be fine-tuned with a small number of training samples for another environment. More detailed definitions and explanations were presented in the class PowerPoint slides.

Summary of the Performed Simulation:

Since the original code of the paper was not available, and I only had access to its dataset (from the University of Gwent website or the author's GitHub), I simulated part of the paper as follows:

- I trained a deep neural network with the structure (12, 150, 100, 50, 25, 1), according to the paper, to classify environments into LOS or NLOS.
- The concepts of LOS and NLOS had already been presented in class (slide 4 of the PowerPoint).
- A part of the dataset needed for this task was available in the folder features_IIoT_19 as a CSV file.

Model Structue:

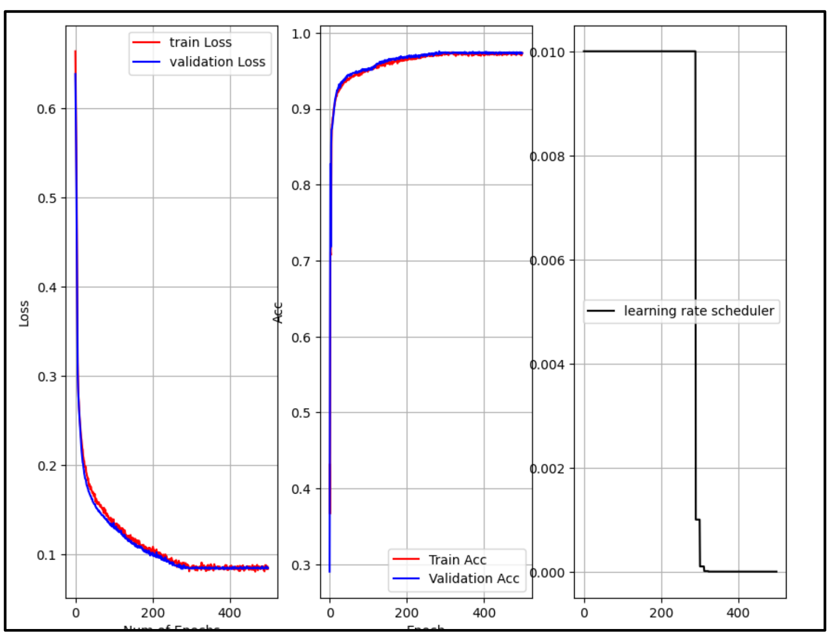
```
num_feats = 12
mum_class = 1

model = nn.Sequential(nn.Linear(num_feats,150),
                      nn.ReLU(),
                      nn.Dropout(0.25),
                      nn.BatchNorm1d(150),
                      nn.Linear(150, 100),
                      nn.ReLU(),
                      nn.Dropout(0.25),
                      nn.Linear(100, 50),
                      nn.ReLU(),
                      nn.Dropout(0.1),
                      nn.Linear(50, 25),
                      nn.ReLU(),
                      nn.Linear(25, 1))

print([sum(p.numel() for p in model.parameters() if p.requires_grad)])
model(x_batch).shape
```

```
[23701]
```

Model Parameters:

- Total parameters: **23,701**
- After applying **PCA** on the input features, the parameters were reduced to about one-third.
- The network was trained on an **RTX 3050 GPU** for 500 epochs (64 minutes).

The following figures show the error correction curves, optimizer learning rate adjustment, recorded accuracy, and training/validation loss during 500 epochs with a learning rate of 0.01.



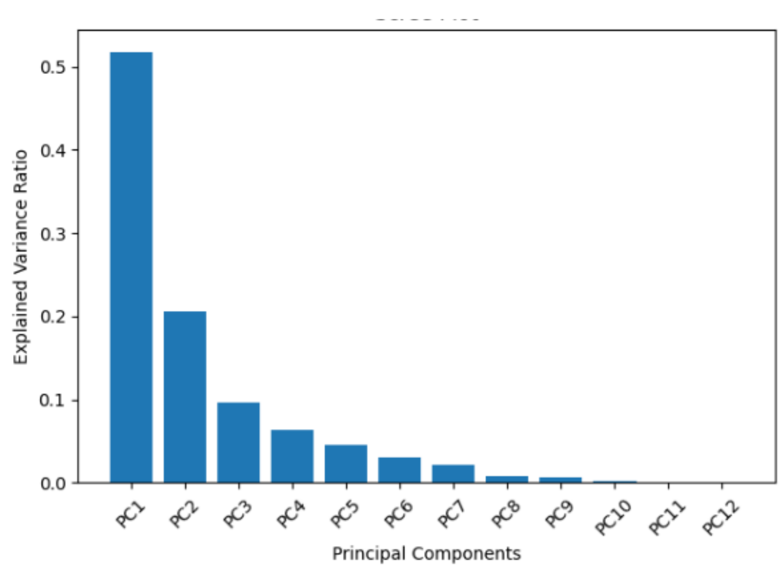
Final Accuracy:

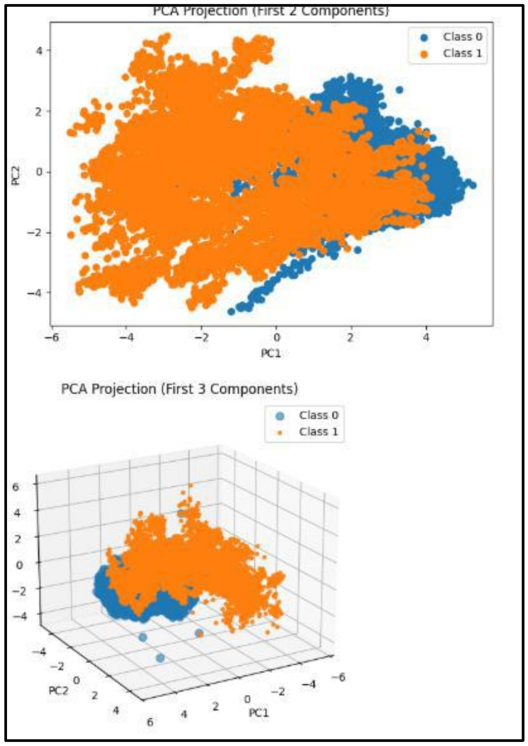
- Accuracy on test data: **98.3%**
- Results are available in the file **DNN-iolab19.ipynb** (inside the “codes” folder).

Improvement Over the Original Method:

Using **PCA**, I examined the possibility of reducing the input feature dimensions:

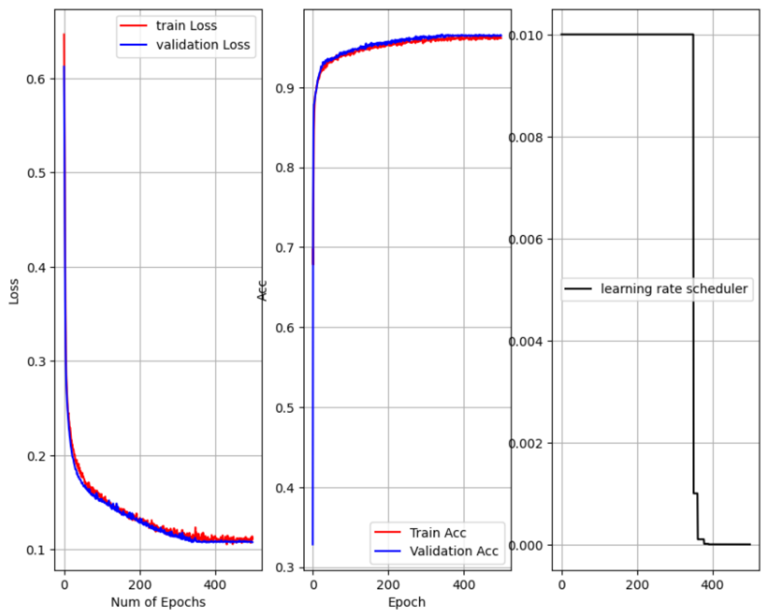
- By calling sklearn.decomposition.PCA on the 12 features, I calculated the variance across PCs 1 to 12.
- Results suggested that inputs could be reduced from 12 to **10 or even 8 features**, decreasing network complexity.
- A scatter plot of the data along PC1 and PC2 showed that separation into LOS and NLOS classes is still possible even after dimensionality reduction.





Reduced Model:

- New structure: **(8, 100, 50, 25, 1)**
- Total parameters: **7,451** (about one-third of the original).
- Trained for 500 epochs.
- Final test accuracy: **96.7%** (only 1.3% lower than the original).



Conclusion:

Although accuracy dropped slightly (~1.3%), the reduction in parameters (to one-third) significantly lightens the network, which can justify the trade-off.