Deep Learning Applications: week 5

Project: Predicting House Prices (Works as a group of 3)

Framework:

- Problem: Accurately predict the selling price of houses in Ames based on historical data.
- Goal: Develop a deep learning model using PyTorch that excels in predicting house prices, outperforming benchmark models.
- Application: This model can empower individuals to estimate their home's value, real estate agents to provide informed pricing recommendations, and investors to make data-driven decisions.

Data Acquisition and Exploration:

1. Download & Import:
   - Obtain the Ames Housing Dataset from www.kaggle.com/ .
   - Use pandas to load the data into a DataFrame.
2. Data Cleaning & Preprocessing:
   - Handle missing values appropriately.
   - Address outliers carefully, considering domain knowledge.
   - Encode categorical features using one-hot encoding or label encoding.
   - Normalize or standardize numerical features to ensure equal importance in the model.
   - Explore feature relationships (e.g., correlations, visualizations) to gain insights.
3. Train-Validation-Test Split:
   - Divide the data into 70% training, 15% validation, and 15% testing sets using sklearn.model_selection.train_test_split.
   - This ensures robust model validation and generalizability evaluation.

Model Development and Training:

1. Model Selection:
   - Consider various architectures based on data characteristics and complexity:
     - Multi-Layer Perceptron (MLP): Good baseline for regression tasks.
     - ~~Convolutional Neural Network (CNN): Can extract spatial features if images are used.~~
     - ~~Recurrent Neural Network (RNN): Suitable for sequential data or temporal dependencies~~.
   - ~~Experiment with different architectures to find the best fit.~~

2. PyTorch Implementation:
    - Start with a basic architecture like MLP, using modules like nn.Linear, nn.ReLU, and nn.MSELoss.
    - Refer to the PyTorch tutorial (https://pytorch.org/tutorials/beginner/basics/quickstart_tutorial.html) for guidance.
    - Define a forward pass to process input data and make predictions.
    - Create a training loop that iterates over batches, calculates loss, uses an optimizer (e.g., Adam) to update model weights, and tracks validation performance.
3. Hyperparameter Tuning:
    - Experiment with hyperparameters like learning rate, batch size, epochs, and other architecture-specific parameters.
    - ~~Use validation set performance to guide tuning and prevent overfitting.~~
    - ~~Consider techniques like grid search or randomized search for efficiency.~~
4. Visualization & Analysis:
    - After each training session, visualize loss curves and validation metrics to track progress and identify potential issues.
    - Visualize feature importance or decision boundaries to understand the model's behaviour.

Fine-tuning and Evaluation:

1. Advanced Techniques:
    - If needed, explore more advanced techniques like data augmentation, regularization (dropout, L1/L2), or ensemble methods to further improve performance.
2. Benchmark Comparison:
    - Compare your model's performance on the testing set with a baseline model (e.g., linear regression) and other machine learning algorithms.
    - ~~Report metrics like MSE, R squared, and MAE to assess accuracy and generalization.~~
3. ~~Error Analysis:~~
    - ~~Analyse errors on the testing set to identify potential biases or shortcomings in the model.~~