Assignment 2: Neural Networks: Advanced

NOTE: This is a group assignment, form group of two or three students and work collectively.

Objective

Implement a complete pipeline for data processing, neural network modeling, and evaluation using Python, scikit-learn, and PyTorch, for following advanced neural network architectures:

1. Convolutional Neural Networks
2. Recurrent Neural Networks
    a. Long Short Term Memory Network (LSTM)
    b. Transformers

Perform the given tasks related to each of the architecture over the corresponding given datasets.

Task 1.1: Dataset Loading and Preprocessing

Datasets

1. **Image Classification and Segmentation: Oxford-IIIT Pet Dataset**
    - Dataset: Oxford-IIIT Pet dataset
    - Task: Predict the median value of owner-occupied homes
    - Features: 37 category pet dataset with roughly 200 images for each class.
    - Output: Predicted class of each image and corresponding mask.
    - Link: https://www.robots.ox.ac.uk/~vgg/data/pets/
2. **Temperature Prediction**
    - Dataset: Daily Minimum Temperature in Melbourne dataset
    - Task: Predict the minimum temperature for different dates in Melbourne.
    - Features: Date, and daily minimum temperature from 1981-1990.
    - Output: Temperature
    - Link: kaggle.com/datasets/paulbrabban/daily-minimum-temperatures-in-melbourne

Implement a DataProcess Class

Create a DataProcess class that handles data loading, preprocessing, and analysis for both datasets.

Oxford-IIIT Pet Dataset:

- Resize all images to a uniform size (e.g., 128x128 pixels).
- Normalize pixel values to the range [0, 1].
- For Image Classification:
    - Assign breed labels to images (one label per image).
- For Image Segmentation:
    - Load corresponding segmentation masks.
    - Ensure that the masks have the same dimensions as the images (128x128).
- Provide data insights:
    - Basic statistics (mean, standard deviation, etc.).
    - Feature distributions.
    - Correlation analysis.
- Split the data into training (80%) and testing (20%) datasets.

Daily Minimum Temperature in Melbourne dataset

- Normalize the temperature values to the range [0, 1].

- Create Sliding Windows:
  - Use the past 30 days of temperatures to predict the next day's temperature.
- Split the data into training (80%) and testing (20%) datasets.

Example

```
class DataProcess:
    def __init__(self, dataset_name, image_size, window_size):
        self.dataset_name = dataset_name
        self.image_size = image_size
        self.window_size = window_size
        self.data = None
        self.labels = None
        self.masks = None


    def _load_oxford_pet_data(self):
        """
        Load Oxford-IIIT Pet dataset: images, breed labels, and
        segmentation masks.
        """
        # Load images and masks (use ImageFolder or custom dataset)
    def _preprocess_oxford_pet_data(self):
        """
        Preprocess Oxford Pet data:
        - Resize images.
        - Normalize pixel values.
        - Prepare data for classification and segmentation tasks.
        """
        # Apply transformations to the images and masks
    def _visualize_oxford_pet_data(self):
        """
        Visualize sample images and segmentation masks.
        """
    def _load_melbourne_temperature_data(self):
        """
        Load the Daily Minimum Temperatures in Melbourne dataset.
        """
        # Read the CSV file and extract relevant columns (date and
          temperature).
    def _preprocess_melbourne_temperature_data(self):
        """
        Preprocess the Melbourne temperature data:
          - Normalize temperature values.
          - Create sliding windows for time-series prediction.
        """
        # Create sliding windows and split data into training/testing
    def _visualize_melbourne_temperature_data(self):
        # Visualize the temperature data distribution.
    def _create_sliding_windows(self, data):
        """
        Create sliding windows for temperature prediction using past
        'window_size' days.
        """
```

Task 1.2: Model Definition

Create a CNN class consisting of **any one of following two models**

- **Convolutional Neural Networks (CNNs)**
    - Include methods for convolutional layers, pooling layers, and dense layers.
    - Use a softmax layer for predicting classes.
- **U-Net**
    - Include methods for encoder and decoder.
    -

Create a RNN class consisting of **any two of following three models**

- **Recurrent Neural Networks (RNNs)**
    - Include methods for 2-layer RNN with ReLU or Tanh activation.
    - Use a dense layer for prediction task.
- **Long-Short Term Memory Networks (LSTM)**
    - Include gating mechanisms to handle long-term dependencies.
- **Transformers**
    - Include self-attention for modeling temporal dependencies.

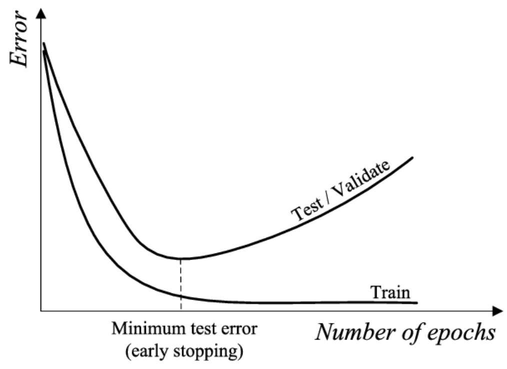
Task 1.3: Training and Evaluation

Use the `DataProcess`, `CNN`, `RNN` classes to train and evaluate models on their respective datasets.

- **CNN:**
    - Use Cross-Entropy Loss and Adam optimizer.
    - Evaluate the model using accuracy on the test set.
- **UNet:**
    - Use Dice Loss or Binary Cross-Entropy Loss.
    - Evaluate using suitable metrics.
    - Visualize segmentation results (input image, ground truth mask, and predicted mask).
- **RNN/LSTM/Transformer**
    - Train the models using MSE Loss and Adam optimizer.
    - Evaluate the models using metrics like Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE).
    - Compare actual vs. predicted temperatures and plot the results.

Submission Guidelines

1. One single Jupyter notebook containing:
    - DataProcess class implementation
    - CNN and RNN class implementation
    - Evaluation metric functions
    - Code for training and evaluating models.
        - Take the hyperparameter values by your choice. All models are to be trained for maximum 20 epochs.
        - Apply early stopping on the basis of test accuracy with patience = 5.
    - Make graph of (train & test) error vs epoch for each model as shown below.



2. In the notebook itself, make one cell at last and report the following,
    - Data insights for both datasets
    - How does the depth of a CNN (number of convolutional layers) affect its performance on image classification tasks? Could adding too many layers degrade performance, and if so, why?
    - In the context of the Oxford-IIIT Pet dataset, how would you determine if the CNN is learning relevant features for classification? What techniques could you use to visualize or interpret the learned features?
    - How does the length of the input sequence impact the ability of an RNN to learn dependencies? What challenges arise when the sequence length increases, and how can these challenges be mitigated in practice?

The format should be followed strictly.
Deadline: November 20, 2024 (Wednesday), 05:00 PM.
No submissions will be entertained after deadline.

All the Best!