Assignment 1

1. Introduction

In this project, the goal is to build a deep learning model that takes an input image (containing multiple digits) and predicts the **sum** of those digits as a **single integer** output. This can be framed as a **classification** problem where each class corresponds to a possible sum (e.g., from 0 to 45 if each image can have up to five digits, each digit 0–9).

We use **PyTorch** to implement a Convolutional Neural Network (CNN), train it on a labeled dataset, and then perform inference on unseen validation data.

2. Dataset and Preprocessing

- **Data**: The dataset consists of NumPy arrays (`.npy` files). Each file contains grayscale images of digits, and a corresponding label file contains the numeric sum for each image.
- **Shapes**: Each image is of size (H,W). For example, images might be 28×28 pixels if dealing with standard digit images (though other shapes are possible).

  (H,W)(H, W)
- **Normalization**: Images are typically converted to `float32` and normalized to the range [0,1] by dividing by 255.0.

  [0,1][0,1]
- **Splitting**: The data is split into a **training** set and a **validation** set, typically with an 80/20 ratio.

3. Model Architecture (Training Notebook)

1. **Convolutional Layers**: Two convolution blocks (`Conv2d + ReLU + MaxPool2d`) to extract features from digit images.

2. **Flattening**: After the final pooling layer, the feature map is reshaped into a 1D vector.
3. **Fully Connected Layer(s)**: A `Linear` layer (with ReLU) to further learn a representation, followed by an output `Linear` layer with dimension equal to the number of possible sum classes.
4. **Loss & Optimization**: Used **CrossEntropyLoss** for classification and an **Adam** optimizer with a small learning rate (e.g., 1e-3).

During training, we:

- Loaded batches of training images and labels from a `DataLoader`.
- Forwarded each batch through the CNN to get predictions.
- Computed the **cross-entropy** loss against the true sum labels.
- Performed **backpropagation** with `loss.backward()` and an **optimizer step** to update weights.
- Monitored the **training loss** and **validation accuracy** each epoch.

4. Inference (New Notebook/File)

To perform inference on the **validation set** in a **new environment** (or separate file) **without** rewriting the CNN:

- **Prepare** each validation image by:
  - Converting it to a PyTorch tensor (`torch.from_numpy`),
  - Adding channel (`unsqueeze(0)`) and batch dimensions,
  - Normalizing if necessary (e.g., `/255.0`).
- **Pass** the image to `model_infer(img_tensor)`, take the `argmax` of the output to get the predicted sum class.

- **Compare** the predicted sum with the true label, measure accuracy, or just visualize.