Assignment 2 Nikhil Mhatre 1002122555
Nxm2555@mavs.uta.edu

My solution: https://github.com/ghost9933/LENET-5

About the Data set:

The dataset contains 70,000 grayscale images of handwritten digits (0-9), where each image is 28x28 pixels.

- **Structure**:
  - Training **Set**: 60,000 images.
  - Test Set: 10,000 images.
  - **Labels**: Each image is labeled with the corresponding digit it represents

The data processing involves loading the MNIST training and test data from `.pt` files. The images are padded from 28x28 to 32x32 to match the input size expected by LeNet-5. The padding and normalization are applied using Torchvision's transformations. The dataset is loaded into Data Loader objects for efficient batching and shuffling.

About LENET-5:
It is a CNN classifier which can classify handwritten digit images.

Layer Structure and Design

1. **Input Layer**:
   a. **Input Size**: LeNet-5 accepts 32x32 pixel grayscale images. This size allows for sufficient spatial detail while being manageable for computational resources.
2. Convolutional Layer 1 (C1**)**:
   a. **Parameters**: 6 filters of size 5x5.
   b. **Operation**: Convolution followed by a nonlinear activation function (typically the hyperbolic tangent, tanh).
   c. **Output**: Produces 28x28x6 feature maps.
   d. **Purpose**: The initial convolutional layer detects basic features like edges and textures. The choice of small filters allows the network to capture local patterns effectively.
3. Subsampling Layer 1 (S2**)**:
   a. **Type**: Average pooling (subsampling) with a 2x2 filter and stride of 2.
   b. **Output**: Reduces the dimensions to 14x14x6.

    c. **Purpose**: This layer reduces the spatial size of the representation, decreasing computational complexity and helping to make the features more invariant to small translations.

4. Convolutional Layer 2 (C3**)**:
    a. **Parameters**: 16 filters of size 5x5.
    b. **Operation**: Only some of the filters connect to the outputs from the previous layer (to reduce redundancy).
    c. **Output**: Produces 10 feature maps (though the design allows for 16, only 10 are connected).
    d. **Purpose**: This layer learns more complex features from the outputs of the first pooling layer, and the selective connectivity helps in reducing overfitting.
5. Subsampling Layer 2 (S4**)**:
    a. **Type**: Average pooling with a 2x2 filter and stride of 2.
    b. **Output**: Reduces the size to 5x5x16.
    c. **Purpose**: Like S2, this pooling layer further reduces the spatial dimensions while retaining essential information, contributing to translational invariance.
6. **Fully Connected Layer 1 (C5)**:
    a. **Parameters**: 120 neurons, fully connected to the previous layer's outputs.
    b. **Purpose**: This layer acts as a transition from convolutional to fully connected layers, consolidating learned features into a fixed-size representation that can be used for classification.
7. **Fully Connected Layer 2 (F6)**:
    a. **Parameters**: 84 neurons.
    b. **Purpose**: It serves to further refine the representation produced by the C5 layer, making it more amenable for classification.
8. **Output Layer**:
    a. **Parameters**: 10 output neurons (for each digit from 0 to 9).
    b. Activation Function: SoftMax function is typically used here to convert the raw scores into probabilities for each digit.
    c. **Purpose**: This final layer produces the predicted class for the input image based on the learned representations.

How it Works

- **Hierarchical Feature Learning**: The network to learn features hierarchically starting from simple edges in the early layers to complex patterns in the later layers.

- **Parameter Efficiency**: The use of filters and breaking the input image into subparts using the convolutional layers and using the pooling layers to reduce the dimension results in a low overall parameter count.
- **Invariance**: Pooling layers reduce the spatial dimensions of feature maps by summarizing the outputs from the input, thing generalizing the input and allows the model to focus on the import features of the input

CODE:

- `data.py`: Contains dataset and data loader functions for MNIST.
- `LeNet5.py`: Defines the LeNet-5 model architecture.
- `train.py`: Contains the training function for the model.
- `eval.py`: Provides evaluation functions for model accuracy and class-wise performance.
- `helper.py`: functions for saving/loading models and plotting training loss.
- `main.py`: The main script that ties everything together.

Training Process (train.py and main.py)

The training process involves defining the optimizer, loss function, and the training loop. The code uses the Stochastic Gradient Descent (SGD) optimizer with a learning rate of `0.01`, and the Cross-Entropy Loss function, which is standard for classification tasks.

- **Training Loop**: The loop iterates over the training data for a specified number of epochs (`Num epochs=15`). The training script (`train.py`) takes care of backpropagation, weight updates, and printing the running loss at intervals for progress tracking.
- **Model Saving**: At the end of training, the model's state is saved to a file for later use.

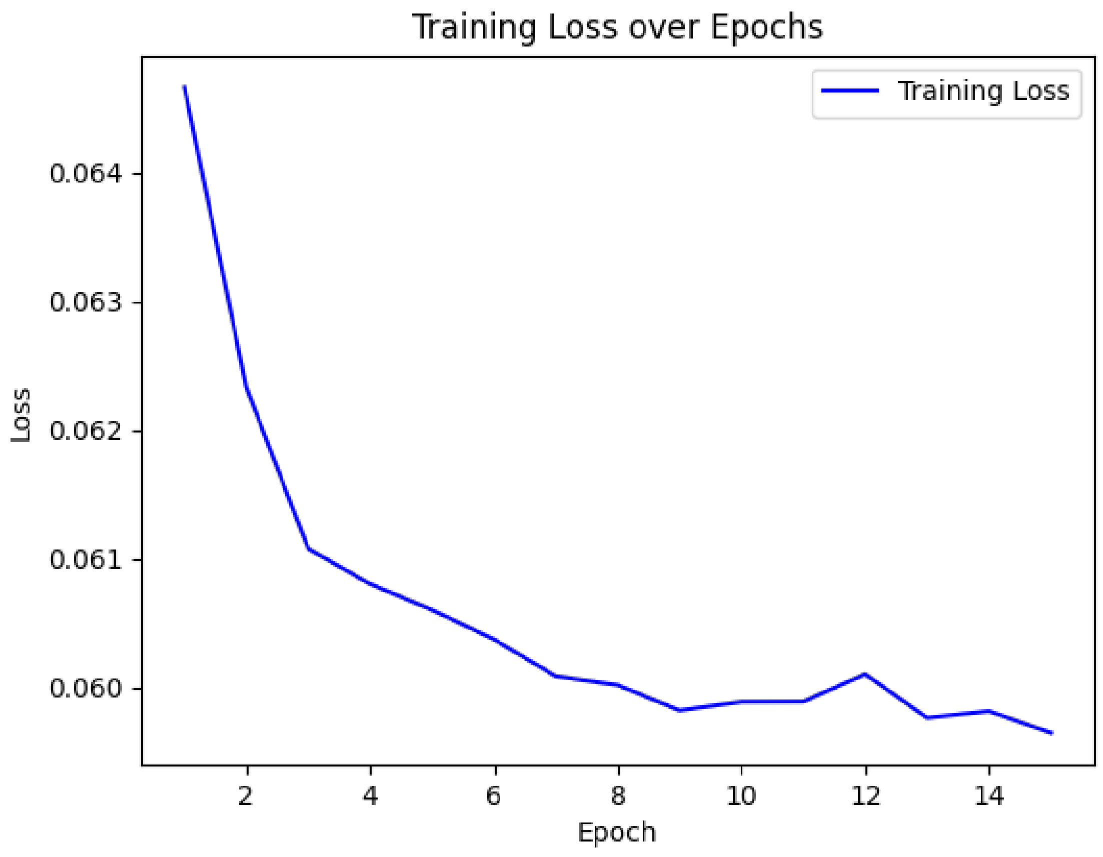
Evaluation (eval.py)

The evaluation script loads the saved model weights and tests its performance on the test dataset. Metrics such as accuracy are computed to understand the model's ability to generalize.

Utility Functions (helper.py)

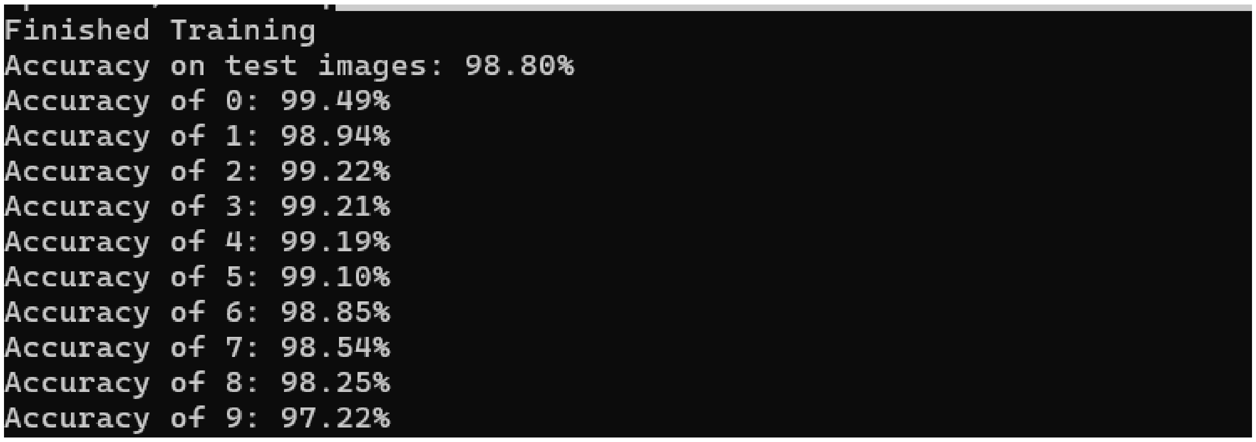
Utility functions are provided for saving and loading the model (`save_model` and `load_model`). Additionally, the script contains functions for plotting the training loss over epochs to visualize model convergence.

Training:



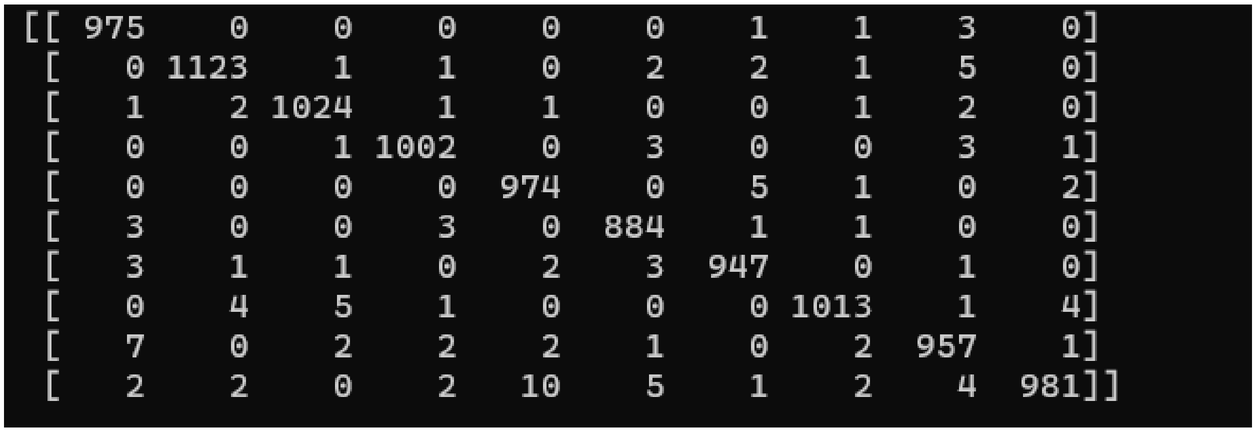
Testing:

Accuracy for the entire test set and classifying each digit in the test set:



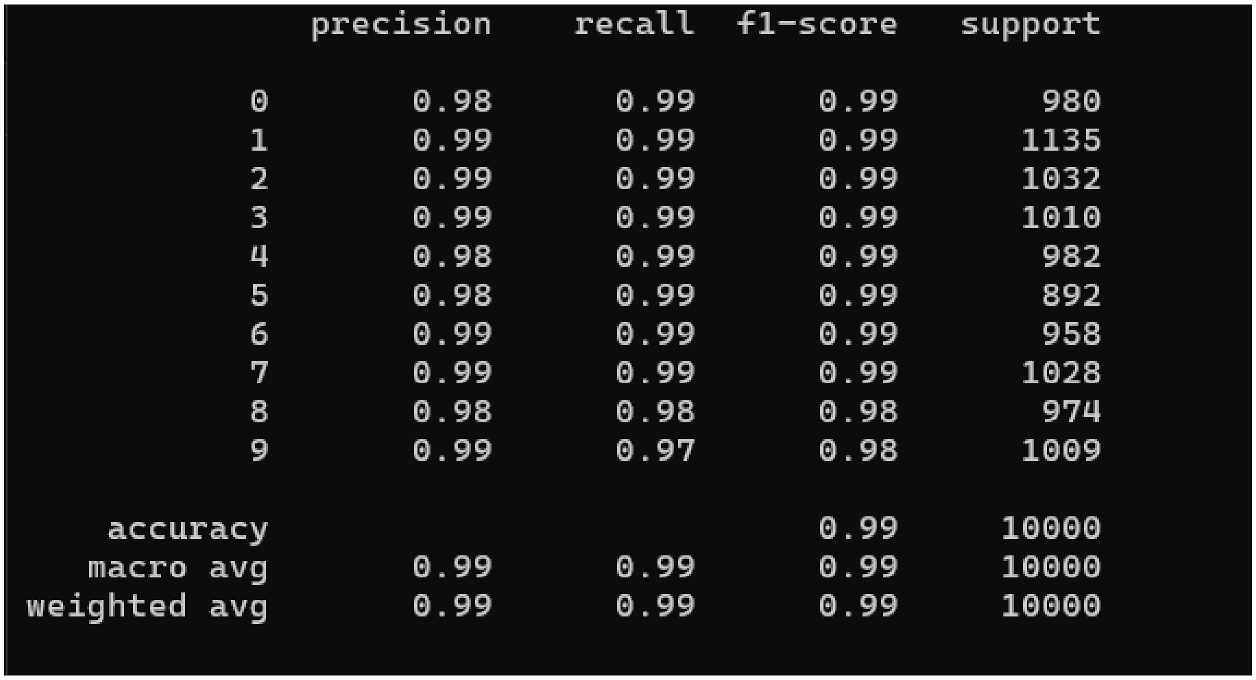
```
Finished Training
Accuracy on test images: 98.80%
Accuracy of 0: 99.49%
Accuracy of 1: 98.94%
Accuracy of 2: 99.22%
Accuracy of 3: 99.21%
Accuracy of 4: 99.19%
Accuracy of 5: 99.10%
Accuracy of 6: 98.85%
Accuracy of 7: 98.54%
Accuracy of 8: 98.25%
Accuracy of 9: 97.22%
```

Confusion Matrix:



```
 [[ 975    0    0    0    0    0    1    1    3    0]
  [    0 1123    1    1    0    2    2    1    5    0]
  [    1    2 1024    1    1    0    0    1    2    0]
  [    0    0    1 1002    0    3    0    0    3    1]
  [    0    0    0    0  974    0    5    1    0    2]
  [    3    0    0    3    0  884    1    1    0    0]
  [    3    1    1    0    2    3  947    0    1    0]
  [    0    4    5    1    0    0    0 1013    1    4]
  [    7    0    2    2    2    1    0    2  957    1]
  [    2    2    0    2   10    5    1    2    4  981]]
```

Classification report :



```
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       980
           1       0.99      0.99      0.99      1135
           2       0.99      0.99      0.99      1032
           3       0.99      0.99      0.99      1010
           4       0.98      0.99      0.99       982
           5       0.98      0.99      0.99       892
           6       0.99      0.99      0.99       958
           7       0.99      0.99      0.99      1028
           8       0.98      0.98      0.98       974
           9       0.99      0.97      0.98      1009

    accuracy                           0.99     10000
   macro avg       0.99      0.99      0.99     10000
weighted avg       0.99      0.99      0.99     10000
```

References :

1. https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62
2. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html
3. https://towardsdatascience.com/pooling-layers-in-convolutional-neural-networks-843d177fb051
4. **https://medium.com**/analytics-vidhya/convolutional-**neural-networks-and**-their-working-8ebd8c8e9b44
5. **https://www.digitalocean.com/community/tutorials/an**-intuitive-guide-to-convolutional-neural-networks
6. https://www.educative.io/blog/convolutional-neural-networks
7. https://www.digitalocean.com/community/tutorials/building-a-convolutional-neural-network-from-scratch-using-pytorch
8. https://www.datasciencecentral.com/lenet-5-a-classic-cnn-architecture/