# CS461 Homework 4: LeNet-5 Implementation and Modifications

Joshua Park (mp1781), Kyle Lee (khl48), Mannan Shukla (ms3389)
CS461: Machine Learning Principles

May 6, 2025

## 1 Problem 1: Implementation of LeNet-5

### 1.1 Architecture

To generate the 7×12 bitmap images for each digit, we followed these steps:

1. **Grayscale Conversion:** The original 128×128 images were converted to a single output channel, reducing them to grayscale to simplify processing.

2. **Resizing:** Each image was resized to the target dimensions of 12×7 (12 pixels in height and 7 pixels in width) using interpolation. This resizing step reduces computational complexity while retaining the key features of each digit. It also allows these to be embedded as fixed parameters in the RBF output layer by ensuring the flattened size of these images is equal to the size of the RBF layer input size.

3. **Tensor Conversion:** The resized images were converted into PyTorch tensors to facilitate further processing and integration into the training pipeline.

4. **Intensity Adjustment:** To enhance the visibility of non-zero pixels and reduce blur, the pixel intensity values were scaled by a factor of 1.5.

5. **Clipping:** Any pixel values exceeding 255 after intensity scaling were clipped (truncated) to the maximum allowable value of 255, ensuring that the images remained within the valid range for grayscale representation.

This preprocessing pipeline ensured that the resulting 7×12 bitmaps effectively retained the critical structural features that are common for each class of digits while matching the input size of the RBF layer.

In the original LeNet5 paper, LeCun et al. argued that the RBF kernel with these fixed model digit parameters necessitated that LeNet-5 explicitly learned to classify digits, and not other unrelated non-digits.

The weights are available at:
https://rutgers.box.com/s/3mzec352bpjpnw96rkm4l7czqn0raj9h

## 1.2 Training Process

The file data.py is attached with this paper. It is necessary to run this file first because the data is not attached.

If one does not have the required packages, use the following commands to create a virtual environment with the necessary packages.

```
python -m venv venv
source venv/bin/activate
pip install -r requirements.txt
```

Then, run data.py to download and preprocess data.

```
python data.py
```

Finally, run train1.py to train the problem 1 model.

```
python train1.py
```

## 1.3 Performance Evaluation

Our train performance at epoch 20 was 0.983988333.... Our test performance at epoch 20 was 0.9875.

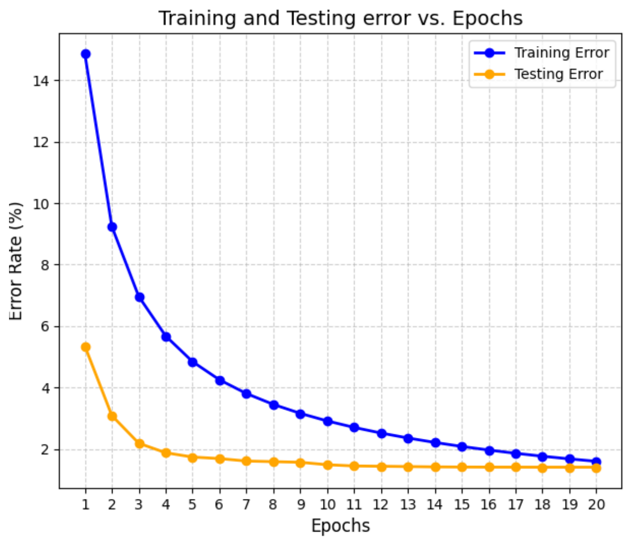


Figure 1: Training and test error of LeNet-5 as a function of the number of passes through the training set for problem 1.

The most confusing for 0 was 6.
The most confusing for 1 was 7.
The most confusing for 2 was 7.
The most confusing for 3 was 5.

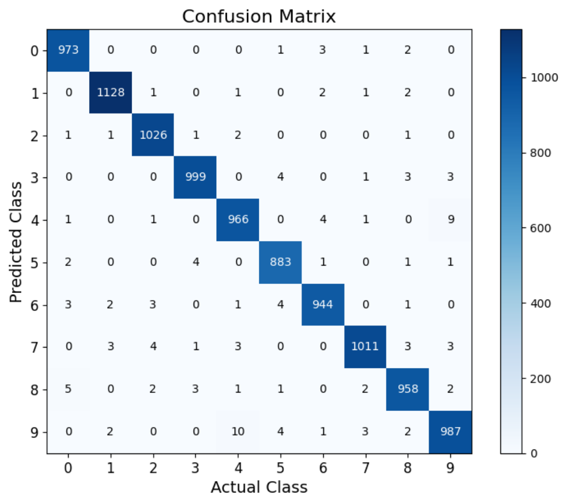
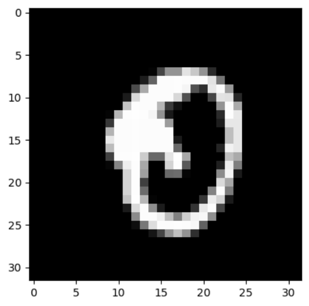


Figure 2: Confusion matrix for problem 1 test



The most confusing for 4 was 9.
The most confusing for 5 was 3 and 9, tied.
The most confusing for 6 was 4.
The most confusing for 7 was 9.
The most confusing for 8 was 3 and 7, tied.
The most confusing for 9 was 4.

There are many symmetrices in the confusion matrix. For example, 4 and 9 both get confused by each other, which is reasonable given their shapes.

To run this test, include LeNet5_1.nt in the same directory as test1.py, and then run test1.py. Note that the virtual environment is still necessary as in problem 1.2 if the required libraries are not installed.

```
python test1.py
```

The most confusing image for 0 was, which was predicted as 8
The most confusing image for 2 was, which was predicted as 0
The most confusing image for 3 was, which was predicted as 5
The most confusing image for 4 was, which was predicted as 6
The most confusing image for 5 was, which was predicted as 8

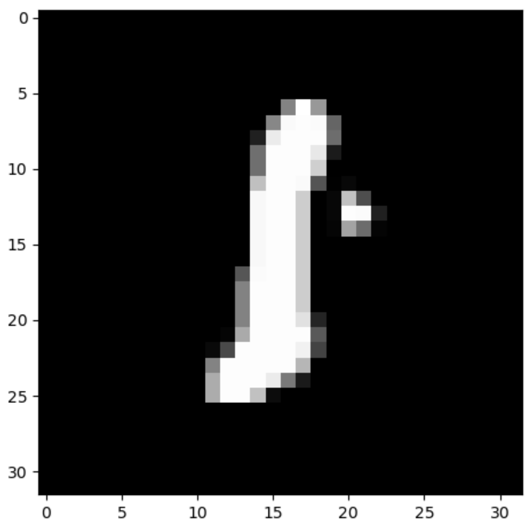
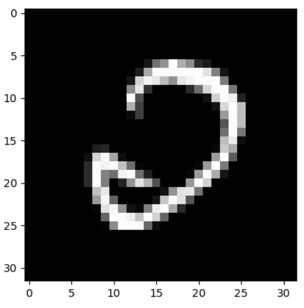
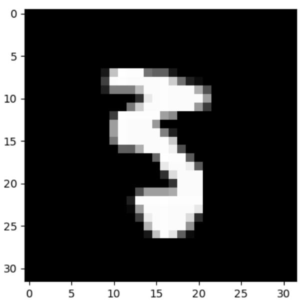
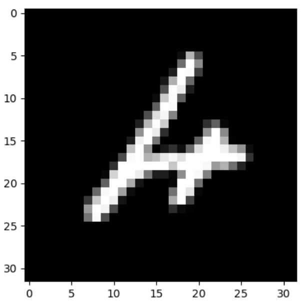


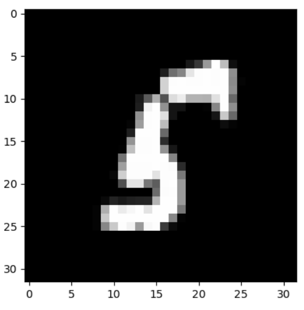
Figure 3: The most confusing image for 1 in Problem 1, predicted as 8.

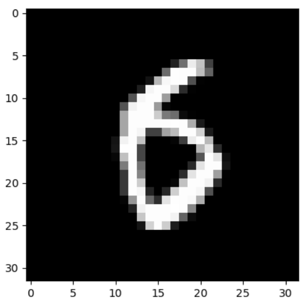
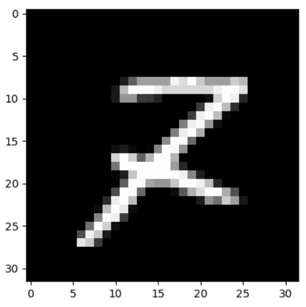
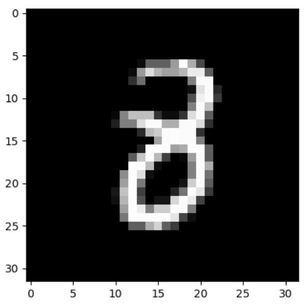
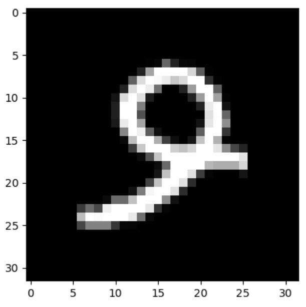
The most confusing image for 6 was, which was predicted as 0
The most confusing image for 7 was, which was predicted as 2
The most confusing image for 8 was, which was predicted as 3
The most confusing image for 9 was, which was predicted as 8


0
5
10
15
20
25
30
0
5
10
15
20
25
30


0
5
10
15
20
25
30
0
5
10
15
20
25
30


0
5
10
15
20
25
30
0
5
10
15
20
25
30


0
5
10
15
20
25
30
0
5
10
15
20
25
30

## 2 Problem 2: Modifications to Handle Unseen Data

### 2.1 Proposed Modifications

We enhance the diversity of the training dataset through augmentation by applying geometric transformations to the `MNIST` dataset in order to better simulate real-world variations in handwritten digits. The transformations applied include:

- **Rotation**: Randomly rotate images within a range of $-30°$ to $30°$.
- **Translation**: Shift images by up to 20
- **Scaling**: Resize images randomly within a range of 80
- **Shearing**: Apply random shearing transformations within a range of $-15°$ to $15°$.

These transformations are implemented using the `transforms.RandomAffine` function in PyTorch.

During training, we switch the hyperbolic tangent activation function with the rectified linear unit as we saw better performance with ReLU. We also used the Adaptive Moment Estimation optimizer (ADAM) as we saw much faster convergence than using shochastic gradient descent. These changes helped us iterate faster and experiment much more with the data, given the significantly low batch size of 1 in the original LeNet-5 paper.

Furthermore, the loss function represented by equation 9 in the original LeNet-5 paper was replaced by Cross Entropy loss, which performs softmax implicitly, thus also adding a softmax layer at the end of the output layer.

Max pooling was considered. However, we found that it degraded the performance of the model, seemingly as it seemed to hide nuances and seemed to not work with the RBF kernel output layer very well.

We also included dropout layer after C5, right before the fully connected layer F6 in the LeNet-5 paper. This is known to help with generalization as the entire network cannot rely on a specific set of neurons.

These changes create a more robust model that can effectively handle the geometric transformations and variations present in the unseen MNIST dataset.

Please note that we used the test split that was given to us in the problem description instead of the announcements. We were unable to find one of the required files that showed labels for the test dataset. This may impact testing. To add a custom dataset, it is possible to change the main() function in test2.py. Any MNIST dataset where __getitem__ is implemented to return a tuple (x, y) will suffice. To run this, please install the requirements and run

```
python test2.py
```