# Fall 2018 IE534/CS598: HW2

**Name**: Ziyu Zhou, **NetID**: ziyuz2

> HW2:
>
> Implement and train a **convolution neural network** from scratch in Python for the MNIST dataset (no PyTorch). You should write your own code for convolutions (e.g., do not use SciPy's convolution function). The convolution network should have a single hidden layer with multiple channels. It should achieve 97-98% accuracy on the Test Set. For full credit, submit via Compass (1) the code and (2) a paragraph (in a PDF document) which states the Test Accuracy and briefly describes the implementation. Due September 14 at 5:00 PM.

## Test accuracy

`98.2%` . See the following training and testing output:

```
loading dataset...
training model...
hyperparameters:
             learning rate = 0.001
             epochs = 40
             filter size = 7
             num of channels = 32
epoch 1, training accuracy = 0.9070833333333334
--- 238.11336016654968 seconds ---
epoch 2, training accuracy = 0.95065
--- 238.03518319129944 seconds ---
epoch 3, training accuracy = 0.9629166666666666
--- 235.76049184799194 seconds ---
epoch 4, training accuracy = 0.9696833333333333
--- 236.4915680885315 seconds ---
epoch 5, training accuracy = 0.9731166666666666
```

```
--- 234.53484416007996 seconds ---
epoch 6, training accuracy = 0.9723333333333334
--- 234.613853931427 seconds ---
epoch 7, training accuracy = 0.9764333333333334
--- 232.31194496154785 seconds ---
epoch 8, training accuracy = 0.9794833333333334
--- 233.63616299629211 seconds ---
epoch 9, training accuracy = 0.9801
--- 240.53144073486328 seconds ---
epoch 10, training accuracy = 0.9809666666666667
--- 235.9505319595337 seconds ---
epoch 11, training accuracy = 0.9833166666666666
--- 234.40735983848572 seconds ---
epoch 12, training accuracy = 0.9830333333333333
--- 235.76307797431946 seconds ---
epoch 13, training accuracy = 0.9847
--- 238.93752908706665 seconds ---
epoch 14, training accuracy = 0.9845166666666667
--- 236.73591589927673 seconds ---
epoch 15, training accuracy = 0.98535
--- 234.08607506752014 seconds ---
epoch 16, training accuracy = 0.9856833333333334
--- 235.52070999145508 seconds ---
epoch 17, training accuracy = 0.9864166666666667
--- 239.65490889549255 seconds ---
epoch 18, training accuracy = 0.9882666666666666
--- 234.72160530090332 seconds ---
epoch 19, training accuracy = 0.9873666666666666
--- 236.72055006027222 seconds ---
epoch 20, training accuracy = 0.98855
--- 252.20952486991882 seconds ---
epoch 21, training accuracy = 0.9896666666666667
--- 239.13087606430054 seconds ---
epoch 22, training accuracy = 0.9903
--- 239.48447108268738 seconds ---
epoch 23, training accuracy = 0.9884666666666667
--- 234.95062899589539 seconds ---
epoch 24, training accuracy = 0.98955
--- 233.94396114349365 seconds ---
epoch 25, training accuracy = 0.9890333333333333
```

```
--- 233.1986198425293 seconds ---
epoch 26, training accuracy = 0.9896666666666667
--- 235.1310589313507 seconds ---
epoch 27, training accuracy = 0.9907333333333334
--- 236.03688406944275 seconds ---
epoch 28, training accuracy = 0.9911
--- 246.92584586143494 seconds ---
epoch 29, training accuracy = 0.9918333333333333
--- 232.69434809684753 seconds ---
epoch 30, training accuracy = 0.99175
--- 230.9488480091095 seconds ---
epoch 31, training accuracy = 0.9922
--- 236.25024580955505 seconds ---
epoch 32, training accuracy = 0.9928166666666667
--- 233.7092990875244 seconds ---
epoch 33, training accuracy = 0.9934666666666667
--- 232.71142601966858 seconds ---
epoch 34, training accuracy = 0.9934666666666667
--- 231.88842391967773 seconds ---
epoch 35, training accuracy = 0.9942666666666666
--- 233.06101989746094 seconds ---
epoch 36, training accuracy = 0.9943
--- 233.8651351928711 seconds ---
epoch 37, training accuracy = 0.9947666666666667
--- 236.6243679523468 seconds ---
epoch 38, training accuracy = 0.9950666666666667
--- 238.33203291893005 seconds ---
epoch 39, training accuracy = 0.99545
--- 235.75587630271912 seconds ---
epoch 40, training accuracy = 0.9958833333333333
--- 235.6646809577942 seconds ---
testing model...
testing accuracy = 0.982
```

## Hyperparameters

The hyperparameters are configured as:

| Hyperparameters | Value |
|---|---|
| Learning rate | 0.001 |
| Number of epochs | 40 |
| Filter size | 7 |
| Number of channels | 32 |

## Usage

Type `python3 main.py` in terminal.

> Note that the default path for the dataset is `"data/MNISTdata.hdf5"` . If a different path is used, please change the input path to the `load_data` function in `main.py` .

To run in debug mode which only trains the model using a smaller part of the dataset, set the `DEBUG` variable to `1` .

## Implementation

The implementation is separated into five files, namely:

- `main.py` : the main file to execute, which contains the high level pipeline of the overall implementation, including configuring the hyperparameters, loading the dataset, initializing the model, training and testing.
- `model.py` : contains the architecture of the Convolutional Neural Network with one hidden layer and multiple channels. The model is implemented as a `ConvolutionalNeuralNetwork` class which supports weight initialization, training and testing. There are mainly two public functions that can be called by the `ConvolutionalNeuralNetwork` object:
    - `train` : train the CNN on the training dataset using SGD.
    - `test` : test the trained model on the testing dataset.

The other functions, i.e., `_forward_step` , `_backward_step` , `_update_weights` , `_predict` and `_reshape_x_to_matrix` are private functions which help with the training and testing process. For more details, please refer to the code docstrings.

- `convolve.py` : contains a `ConvolveOps` class which implements tools to perform convolution operations. To perform convolution, call the `convolve` method. Two kinds of convolution operations have been implemented:
    - `_convolve_brute_force` : brute force convolution operation using for loops
    - `_convolve_optimize` : optimized convolution operation using matrix multiplication whic achieved a speedup of ~20 times.
- `io_tools.py` : contains tools to load the MNIST dataset.
- `activate_functions.py` : implements activation functions for later use, including ReLU and softmax, as well as the gradient for ReLU.