# 1 Outline

In this assignment, you are asked to design MNIST classifier using convolutional neural networks.

# 2 Specification

In this assignment, you are asked to write a Python code to implement **two** neural network classifiers for MNIST dataset. Specifically,

- CNN from scratch: your custom module based on previous homeworks. Implement `forward` method for class `nn_mnist_classifier`. You also need to complete all the layers in nn_layers_pt.py file.
- CNN using PyTorch modules. Complete class `MNISTClassifier_PT` in the file.

The implementation details are provided in the following sections.

# 3 Convolutional Neural Network (CNN) for MNIST

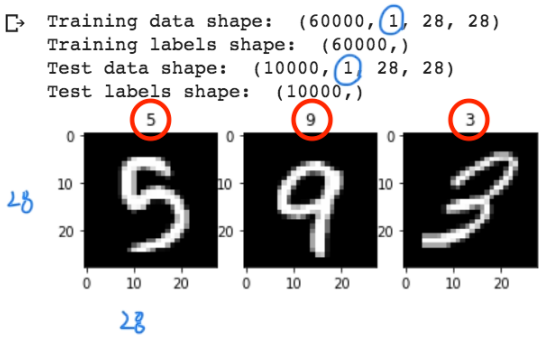


Figure 1: Example of MNIST images.

## 3.1 MNIST dataset

MNIST dataset is a database of images of handwritten digits. There are handwritten numbers of 0 to 9. The dataset contains 60,000 training images and 10,000 test images. The goal of a classifier is to classify the digits correctly to numbers 0 to 9. So this is

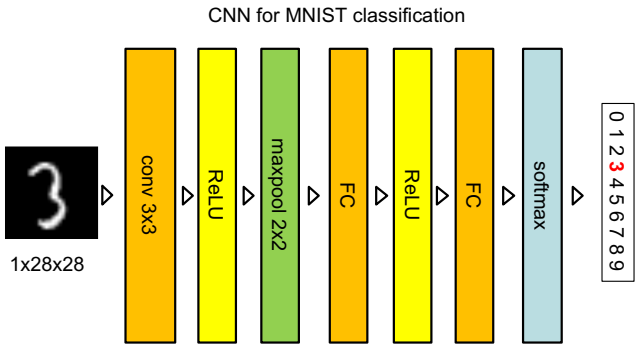


Figure 2: CNN architecture.

a classification problem of 10 classes (0 to 9). The size of single image is 28-by-28 pixels. The image is grayscale, so there is only one color channel. Therefore, an image has dimension 1x28x28.

The size of single image is 28-by-28 pixels. The image is grayscale, so there is only one color channel (unlike RGB three-channels color images from CIFAR-10). In the `hw5.py` file, there is a code which downloads and extracts the training and test dataset. It looks as

```
from keras.datasets import mnist
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

We create four variables, `X_train, y_train, X_test, y_test`.

1. `X_train` : contains the handwritten images, and has shape (60000,1,28,28). The meanings are: 60000 is the number of images, 1 is the number of channels (grayscale means only 1 channel), 28 is width and 28 is height of the image.

2. `y_train` : has shape (60000,). The meanings are: 60000 is the number of images. This data contains the **ground truth labels** of the corresponding images, and contains the number between 0 and 9. For example, if `X_train[0]` contains a handwritten image for number 5, then `y_train[0]` is 5.

3. `X_test` : has shape (10000,1,28,28). This contains the handwritten images for testing.

4. `y_test` : has shape (10000,). This contains the ground truth labels for testing images.

## 3.2 Network Architecture

The proposed CNN is given by Fig. 2. It consists of the following layers in order:

1. Convolutional layer: takes a batch of MNIST images as input. Input $x$ will have shape

   `x.shape=(batch_size, input_channel_size, in_width, in_height)`

   where `input_channel_size`=1, `in_width=in_height`=28. The convolutional layer has 28 filters. Thus the output `y` will have shape

   `y.shape=(batch_size, num_filters, out_width, out_height)`

   where `num_filters`=28, `out_width=out_height`=26.

   This layer is an object of

   - class `nn_softmax_layer` for custom classifier
   - class `torch.nn.Conv2d` for PyTorch classifier

2. Activation layer: ReLU activation is used. This layer is an object of

   - class `nn_activation_layer` for custom classifier
   - class `torch.nn.ReLU` for PyTorch classifier

3. Maxpool layer: A 2x2 maxpooling is done with stride 2. Thus the output `y` will have shape

   `y.shape=(batch_size, 28, 13, 13)`

   - class `nn_max_pooling_layer` for custom classifier
   - class `torch.nn.MaxPool2d` for PyTorch classifier

4. Fully Connected (FC) layer: A linear layer producing linear scores, $y = Wx+b$, where the output channel size is 128. As input, the layer takes $(batch\_size, 28, 13, 13)$. Note however, for each input in the batch, FC layer treats each input whose 3-D volume is $28 \times 13 \times 13$ as a 'flat' vector. Thus, FC layer will regard the input as $(batch\_size, 28 * 13 * 13)$, take inner product with each flattened vector, and produce output `y` with shape

   `y.shape=(batch_size, 128)`

   - class `nn_fc_layer` for custom classifier
   - class `torch.nn.Linear` for PyTorch classifier

5. Activation layer: ReLU activation is used.

   - class `nn_activation_layer` for custom classifier
   - class `torch.nn.ReLU` for PyTorch classifier

6. Fully Connected (FC) layer: Another FC layer, takes input size of 128, and produces output size of 10. These 10 numbers represent the linear scores for each category, that is, from number 0 to 9. The output `y` has shape

   `y.shape=(batch_size, 10)`

- class `nn_fc_layer` for custom classifier
- class `torch.nn.Linear` for PyTorch classifier

7. softmax layer: Softmax output, representing the probability scores for 10 categories, that is, from number 0 to 9. The output `y` has shape

   ```
   y.shape=(batch_size, 10)
   ```

   This layer is an object of

   - class `nn_softmax_layer` for custom classifier
   - You don't need to implement this layer for PyTorch classifier

8. The loss function is the cross-entropy loss. For custom classifier, `nn_cross_entropy_layer` is added at the end of the CNN. For PyTorch classifier, you will use `torch.nn.CrossEntropyLoss` module.

## 3.3 class `nn_mnist_classifier`

Implement method `forward` or Q1 in `hw5.py`.

### 3.3.1 `forward` method

This method performs the forward pass of a batch of MNIST images; that is, the batch is passed through the CNN. The method returns the scores for the batch, and the average cross-entropy loss over the batch.

`nn_mnist_classifier.forward(x,y)`: Performs a forward pass of the classifier. Input parameters are as follows:

**Parameters:**

- $x$: *nd-array*.

  4-dimensional `numpy.array` input map to the convolutional layer. Each dimension of $x$ has meaning

  ```
  x.shape=(batch_size, input_channel_size, in_width, in_height)
  ```

  For example, if `x.shape` returns $(16, 3, 32, 32)$, it means that `x` is an image/activation map of size $32 \times 32$, with three input channels (like RGB), and there are 16 of them treated as one batch (as in mini-batch SGD).

- `y`: *nd-array*

  Ground Truth (GT) labels for input image batch `x`.

  ```
  y.shape=(batch_size,)
  ```

**Returns:** two values `score, loss`.

- `score`: *ndarray*

  2-dimensional `numpy.array` with the following shape.

  ```
  score.shape=(batch_size, 10)
  ```

  Softmax output containing scores for 10 categories (numbers from 0 to 9) for the batch.

- `loss`: *float64*

  Cross-entropy loss.

#### 3.3.2 `update_weights` method

`nn_mnist_classifier.update_weights()`: Update weights according to momentum method. The velocity or momentum parameters are defined, stored and updated within the class object.

#### 3.3.3 `__init__` method

Initialization method for the class.
**Parameters:**

- `mmt_friction`: *float64*.

  The friction or $\alpha$ parameter for momentum method. By default, this is set to 0.9.

- `lr`: *float64*.

  learning rate.

### 3.4 class `MNISTClassifier_PT`

Implement class `MNISTClassifier_PT` by importing PyTorch modules. You need to implement the constructor (Q2) and forward method (Q3). Note that class `MNISTClassifier_PT` is implemented up to the last fully connected layer. The softmax and loss is computed by `torch.nn.CrossEntropyLoss`. Look up the documents for details.

## 4 What to submit

You will be given two files `hw5.py` and `nn_layers.py`.

1. **You are asked to complete Q1–Q3 parts in `hw5.py`.**
2. **You are asked to complete all the layers (Q4–Q5) in `nn_layers_pt.py`.**

Submission instructions are

- Submit modified Python file `hw5.py` and `nn_layers_pt.py`.
- Upload your files at Blackboard before deadline. (Please submit the file in time, no late submission will be accepted).

# 5 How your module will be graded

If done correctly, you will be able to achieve accuracy over 90% **within 2 epochs** of training with proper batch size and learning rate. That is, if your CNN gets trained with the whole training dataset (50000 images) for two times, the accuracy of your model will be high enough. **The goal is to achieve over 80% of accuracy under the following condition:**

**Before testing, your model will be trained until one of the following two events occur, whichever comes earlier.**

1. Your model will be trained for **two** epochs.
2. Your model will be trained for **two hours** under the standard Google Colab compute engine environment (CPU).

The reason we put upper limit on the training time is that, if your model uses too many `for` loops in convolutional and maxpool, the training will be very slow. Also we have time limits for grading the homework, so if the training of your model takes excessively long, then we have to stop it at some point.

So before submitting your model, please try to train your model on the standard Google Colab environment, and check whether your model can achieve over 80% accuracy under 2 hours of training.

# 6 Grading

Perfect score is 20 points:

- 10 points if each of two classifiers achieves total accuracy more than 80%.
- 3 points if each classifier achieves total accuracy between 20–80%.
- 1 points if each classifier achieves total accuracy below 20%.
- 0 point if you do not submit the file by deadline.

In the blackboard, you can upload your files as many times as you like, before the deadline. The last uploaded file will be used for grading. After deadline, the submission menu will be closed and you will not be able to make submission.