Pre-trained MLP Model

Image Classification Application- Today, artificial neural networks are used in various critical applications such as classification, pattern recognition, clustering, optimization, etc. **Image classification** is the process of analyzing the image to identify the 'class' of it (or the probability of the image being part of a 'class'). As shown in figure 2, a class is essentially a label, for instance, 'cat', 'dog', and so on.

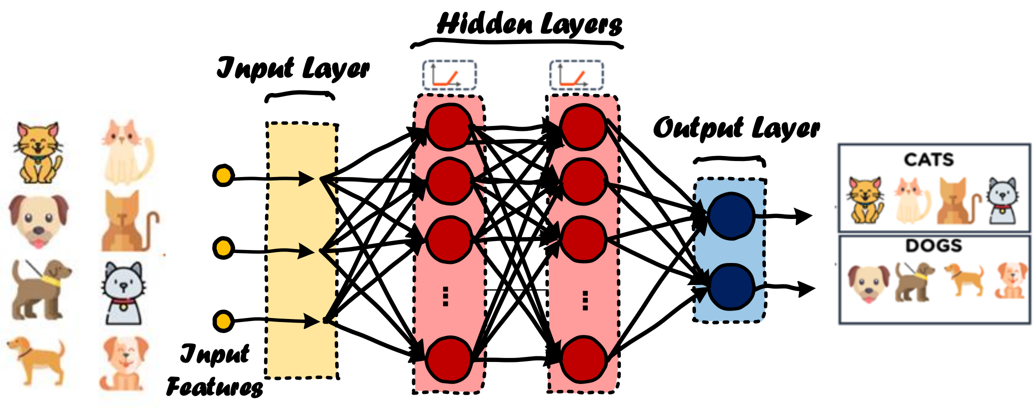

Figure 2. An Example of an image classification application

Input Dataset (MNIST) - In this project, you should implement an MLP network to **classify** a set of images related to the "**MNIST**" dataset that is widely used for training and testing in the field of machine learning. MNIST dataset includes some black and white images of handwritten digits that are between 0 and 9. Some examples of these images are shown in Figure 3.

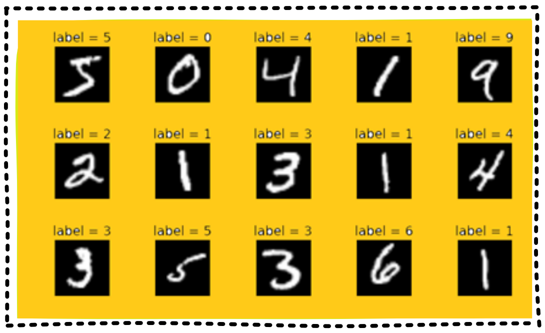

Figure 3. Some example images from the MNIST dataset

- ## The pre-trained neural network

The pre-trained model is a simple MLP neural network that classifies the MNIST dataset. The original MNIST dataset's images have dimensions of 28 x 28 pixels, totally included 784 input features. In order to simplify the classifier neural network, the size of images is reduced to 62 features using some feature reduction algorithms. So, in this case, your neural network will have 62 input features.
MNIST dataset has 10 output classes corresponding to the digits from 0 to 9, in which by recognizing the digit i, the i' th output becomes one. To identify the label of an input image, the maximum value between

the outputs of 10 neurons must be selected. If neuron i has the highest output value, the input image tag will be i.

As shown in Figure 4, the MLP in this project has 62 inputs in the input layer, 10 output neurons in the output layer, and also one hidden layer with 30 neurons is considered for this issue. In both layers of your MLP model for MNIST classification, the ReLU function is considered as an activation function. $\mathrm{ReLU}(x) = \max(0.x)$

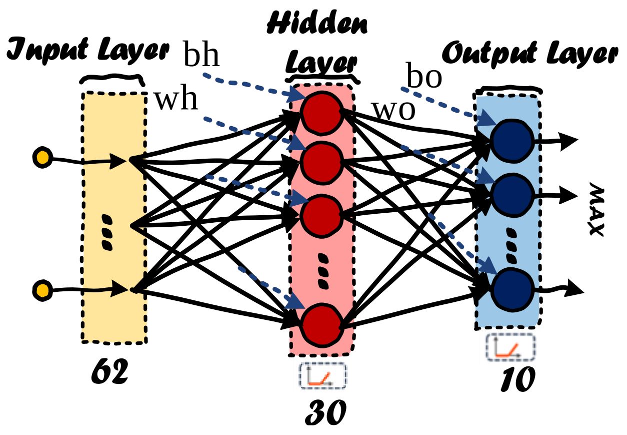

Figure 4. The required MLP network in this project

Note that:

- In order to implement this network, the hidden and output layers' weights and biases are required ($w_{h(\mathrm{hidden})}, w_{o(\mathrm{output})}, b_{h(\mathrm{hidden})}, b_{o(\mathrm{output})}$). Based on the described network, w_h and w_o are two matrices with the dimensions of 30 x 62 and 10 x 30, respectively, including 2160 weights totally. Also, b_h and b_o are matrices (vectors) with dimensions of 30 x 1 and 10 x 1, respectively, including 40 bias values.
- Weights and biases are 8-bit with sign-magnitude representation.
- The values of weights and biases and the network input test data are uploaded in separate files on the course page, which need to be stored in memory according to your design.
- You can take a look at Guideline File in the project folder to see what the content of each file is.
- There are 750 test data in the folder. A testbench should apply at least 100 (out of 750) input to your design. The inputs must be selected randomly from the file.

The pseudo code of this neural network is described below:

Pseudo Code of Neural network:

```
//Initial Parameters
Init Wh[30][62];    //Weights of hidden layer, 8-bit sign-mag
Init Wo[10][30];    //Weights of output layer, 8-bit sign-mag
Init Bh[30];        //Biases of hidden layer, 8-bit sign-mag
Init Bo[10];        //Biases of output layer,8-bit sign-mag


// Get Input feature
Get In[62];         //Input test data, 8-bit sign-mag

// Hidden layer:
H_out [30];
For i=1:30
        H_out[i]=0;
        For j=1:62
                H_out[i] += Wh[i][j]* x In[j];  //Sum of weight-input products
        H_out[i] = (H_out[i] + Bh[i]*127);      //Bias addition
                                                //multiplication is for bias alignment , 127=(01111111)8-bit sign-mag
        H_out[i] = H_out(i) >> 9-bit;           //Shift to right due to scaling
        H_out[i] = ReLU(H_out[i]);              // Activation Function , ReLU=max(H_out[i], 0)
       H_out[i] = Sat(H_out[i]);                //Saturation to 8-bit


// Output Layer;
O_out [10];
For i=1:10
        O_out[i]=0;
        For j=1:30
                O_out[i] += Wo[i][j] x* H_out[j]; //Sum of weight-input products
        O_out[i] = (O_out[i] + Bo[i]*127);                //Bias addition
        H_out[i] = H_out(i) >> 9-bit;             //Shift to right due to scaling
        O_out[i] = ReLU(O_out[i]);                // Activation Function , ReLU=max(O_out[i], 0)
        O_out[i] = Sat(O_out[i]);                 //Saturation to 8-bit


// Softmax
Label = arg max(O_out);                  // calculate the maximum between outputs
```

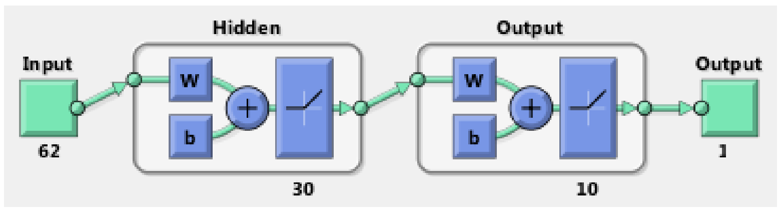

Figure 5. The required MLP network (MATLAB View)