

Handwritten Digit Recognition Using a Convolutional Neural Network and ProMoD Backpropagation

Micah Jeffries

California Polytechnic State University - SLO

mjjeffri@calpoly.edu

Abstract - Convolutional neural networks (CNN) are among the most predominant technology for image pattern recognition. This paper documents the design and experimental results of a handwritten digit recognition system based on a CNN with ProMoD backpropagation. The training data and testing data for this system are taken from the MNIST open source database which contains thousands of handwritten digital images of numerical digits rendered with 28x28 pixels. The system accepts these 28x28 pixel images as inputs and outputs a digit ranging from zero to nine which represents the most probable digit rendered in that image. The deep learning process employs the ProMoD backpropagation algorithm to help expedite the training process.

I. INTRODUCTION

Handwritten digit recognition plays a vital role in many areas of industry from financial banking to government operations. Some of these applications include postal mail sorting, bank check processing, form data entry, etc [1]. This application falls generally into pattern recognition which is a field of study that has been methodically critiqued and improved over the course of time. Not only does automating the process save valuable time, but modern deep learning processes have achieved a higher accuracy of recognition than the traditional method of human observation.

The requirements for the system are to correctly classify individual written digits from zero to nine with a high rate of accuracy. The main impediment to high accuracy is the randomness and personal writing habits of users. To test the accuracy of the system, 24,000 images were obtained from the MNIST open

source database which contains handwritten digital images of numerical digits rendered with 28x28 pixels. 20,000 of these images are used to train the convolutional neural network while the other 4,000 are reserved for testing the accuracy of the system. The images in this database contain a high variation of writing styles and writing quality for each written digit which is representative of the various users who will utilize this technology.

The paper is organized as follows; Section 2 contains a literature review of previous work done to solve this problem. The approach that I took and the elements of previous work which I borrowed from are explained in Section 3. The experimental details and results are presented in Sections 4 and 5. Finally, Section 6 contains some concluding remarks.

II. PREVIOUS WORK

The seemingly simple task of identifying a handwritten number has been approached from many unique angles. One popular approach comes from the idea of feature extraction. In the English numerical system, there are various defining qualities for each digit such as the number of loops [1]. For example, digits 0, 6, and 9 have one loop and digit 8 has two loops.



Figure 1. Top reservoir area of the digit 4 [1]

There is also the water reservoir feature which quantifies the amount of cavity regions in each

digit [1]. For example, if water was dispensed on top of the digit 4, water would be stored in the configuration as shown in Figure 1. Thus, one feature of the digit 4 is that it has a square-shaped cavity region on top of the digit.

This approach, when combined with a simple classifier such as K-Nearest Neighbor, has the benefits of low-time complexity, independence from digit size and writer style, and free of training. However, feature extraction does not often meet the standard of state of the art neural network systems in terms of accuracy [1].

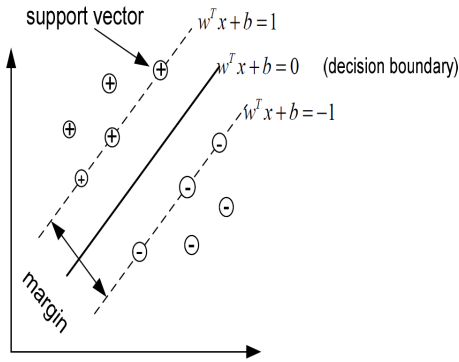


Figure 2. SVM Classification [2]

The accuracy of feature extraction and classification can be improved with the help of SVM (Support Vector Machine). SVM classifies data by deriving the best hyper plane that differentiates all data points of one class from other classes [2]. The best hyper plane maximizes the gap or margin between the two classes as shown in Figure 2. SVM, however, does require training as opposed to a simple K-Nearest Neighbor Classifier.

Another unique approach has been developed by [3] in the form of a digital logic circuit which implements a classification function. The details of a classification function are beyond the scope of this paper, but the reader is invited to read from [3] to learn more about them. The relevance here lies in the fact that a linear circuit consisting of standard digital logic gates such as AND/OR can implement this classification function and identify with reasonable accuracy handwritten digits. Figure 3 presents a simplified diagram of this implementation.

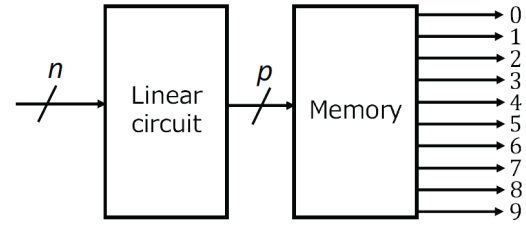


Figure 3. Diagram of Digital logic Circuit Classifier [3]

The accuracy of these circuits derived from the classification function are significantly lower than that of neural networks and other common approaches. However, these digital logic circuits require much smaller hardware and require no learning [3].

The most popular approach to solving this problem comes with the convolutional neural network which naturally accels at image processing applications. A group of researchers at Tongji University proposed a convolutional neural network with two convolution layers, two pooling layers, and a connection layer. The convolution layers extract features from each handwritten digit while the pooling layer reduces the amount of data by truncating the matrices by a factor of four. The connection layer takes this feature extraction and performs classification. The convolution and connection layer in this paper are trained using the gradient descent algorithm [4].

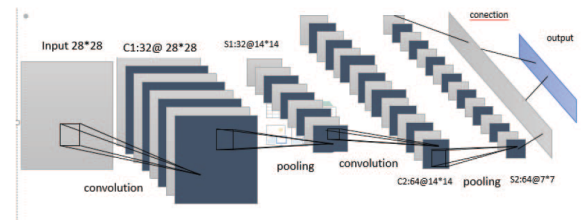


Figure 4. Convolutional Neural Network [4]

This approach has the advantage of high accuracy, however, it requires more complex hardware and has the longest training time over the other approaches. One researcher proposed a strategy to accelerate the convergence time for the gradient descent algorithm by utilizing the ProMoD method [5]. The error is calculated and back propagated to each layer as per the norm. However, when applying the weight correction

to each layer, an additional momentum and derivative correction matrix is applied to the weight correction. This process accelerates the backpropagation algorithm by quickening the convergence of each training layer. This paper also used the concept of training batches to further accelerate the process.

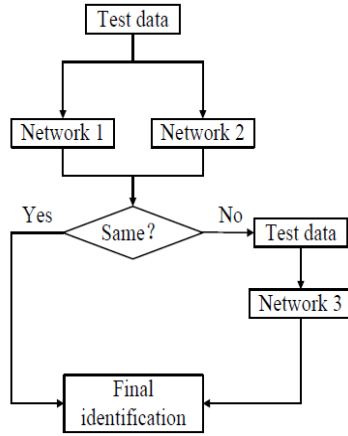


Figure 5. Combined Depth Neural Network [6]

In [6] the researchers combine the convolutional neural network and the Backpropagation (BP) neural network as shown in Figure 5. In the minority of cases where either network incorrectly identifies the digit, the test data is fed into Network 3 to correct the result. This type of architecture can be used to achieve state of the art accuracy at the cost of a more complex network. This paper also implemented the adaptive learning rate adjustment algorithm with equation (1) [6]. The learning rate $n(t)$ adjusts based on the error function $E(t)$. If the learning rate is too large, then the network will not converge. If the learning rate is too small, the training time will take too long.

$$n(t+1) = \begin{cases} 1.05 * n(t) & E(t) < E(t-1) \\ 0.7 * n(t) & E(t) > 1.04 * E(t-1) \\ n(t) & \text{other} \end{cases} \quad (1)$$

One other noteworthy approach to solve this problem was conducted by [7] to explore the efficiency of Self-Organizing Maps (SOM). The SOM is a neural network architecture, similar to the CNN. However, the SOM is trained with unsupervised learning as opposed to the CNN

which is trained with supervised learning. Unsupervised learning is able to train the network with an unlabeled training set. The SOM updates the weights of the neurons in such a way where similar input images are mapped to nearby neurons [7].

Although unsupervised training can be a useful feature to have in cases where training labels are not readily available, it is not as useful for handwritten digit recognition since training sets for handwritten digits are easily and readily available. The SOM architecture also does not handle geometric transformations such as rotations very well. However, it has a very high accuracy and is very effective at dealing with input images with a lot of background noise [7].

III. APPROACH & ALGORITHM

In this paper, the method to solve the problem of handwritten digit recognition will employ a combination of the methods in [4] and [5]. A convolutional neural network with a similar architecture to that shown in Figure 4 is used for this paper. This method has one of the highest recognition rates as compared to all the other methods explained in the previous section. To simplify the process, the number of convolution matrices are reduced which simply reduces the amount of weights that will have to be adjusted. The dimensions of each layer of the network is shown in the table below.

Table 1. Filter Dimensions of Each Layer of the CNN Network

Layer Number	Type	Filter Dimension
1	Convolution	9 x 9 x 4
2	Mean Pooling	2 x 2
3	Convolution	5 x 5 x 8
4	Mean Pooling	2 x 2
5	Connection	10 x 392

The rectified linear unit (ReLU) activation function is used for the convolution layers while the softmax activation function is used for the output layer. Softmax has become a common standard for using in multi-class output layer systems to transform the output results into probabilities as shown in equation (2). The ReLU function is used for the convolution layers because it helps solve the vanishing gradient problem [8]. Sigmoid and hyperbolic tangent functions are non-ideal for deep multi-layer neural networks because they drastically reduce the error as it is back propagated through each layer. The convolution layers deep in the network receive inadequate direction for changing its parameters to reduce the cost function [8]. ReLU alleviates this problem and its calculation is shown in equation (3).

$$f(x_i) = e^{x_i} / \sum_k e^{x_k} \quad (2)$$

$$f(x) = \max(0, x) \quad (3)$$

The ProMoD backpropagation algorithm, as explained in [5], will help to quicken the convergence time of each training layer of the CNN. This will help alleviate the long training time for CNNs. Also, when training the network, the images are divided into batches and the weights are corrected for each batch of images. Instead of correcting the weights for each image, the corrections are accumulated and averaged for a batch of images, then this averaged matrix is applied to correct the weights [5]. This helps alleviate excessive weight correction and reduce training time.

The algorithm for training the CNN with the ProMoD backpropagation algorithm is explained as follows:

1. Load the training images and labels from the MNIST database
2. Divide the training images into batches with a predefined size
3. Randomize the weights for each layer of the network
4. Fetch a training image and feed it into the network to obtain the output y
5. Fetch the corresponding training label and use the desired output d to compute the output error e

$$e = d - y \quad (4)$$

For example, if the training label is 4, then the desired output is $d = [0, 0, 0, 1, 0, 0, 0, 0, 0]$.

6. For each layer k , calculate the delta δ_k by multiplying the error e_k and the derivative of the activation function ψ_k

$$\delta_k = e_k \times \psi_k'(v_k) \quad (5)$$

7. For each layer k , calculate the weight correction Δw_k using the delta δ_k from the previous step and the output y_k

$$\Delta w_k = \delta_k \times y_k \quad (6)$$

8. Repeat steps 4-7 for each image in the training batch and average the weight correction for each layer
9. Obtain the proportional correction P_k , momentum correction I_k , and derivative correction D_k using the current weight correction $\Delta w_k(n)$ and previous weight correction $\Delta w_k(n-1)$ [5].

$$P_k(n) = \Delta w_k(n) \quad (7)$$

$$I_k(n) = I_k(n-1) + \Delta w_k(n) \quad (8)$$

$$D_k(n) = \Delta w_k(n) - \Delta w_k(n-1) \quad (9)$$

10. Update each layer w_k with the correction terms in the previous step [5].

$$w_k = w_k + \alpha \times P_k + \beta \times I_k + \gamma \times D_k \quad (10)$$

11. Repeat steps 4-10 for the number of desired training iterations

Step 7 represents the general algorithm for computing the weight correction for a given layer k . However, there is some nuance to calculating the backpropagation error for the convolution and mean pooling layers. First of all, there are multiple filters in each convolution layer and each one requires its own error correction. Second of all, the weight correction

involves calculations with matrices since these are 2 dimensional filters so the math requires some modification.

Given a single pixel $w_{m,n}$ in the weight kernel at layer k W^k , an input image of X^{k-1} , and the delta kernel δ^k , we need to know the change to the cost function E as a result of a change to the weight pixel. The math here is a little involved, so the derivation is left in the appendix, but the result is shown in equation (11) [9]. Note that the $*$ symbol represents convolution.

$$dE / dw_{m,n} = \text{rot}_{180}(\delta^k_{ij}) * X^{k-1}_{m,n} \quad (11)$$

Similarly, we need a method for computing the error in the pooling layers. No learning takes place in the pooling layers, yet we need to know the delta matrices in this layer because they are required to calculate the weight correction for the previous layers. In backpropagation, the pooling layer is restored to its original size with the new pooling block taking on the error pixel from which it expanded. In the case of an $N \times N$ pooling block, the error in the pooling layer is multiplied by $1 / (N \times N)$ [9].

IV. METHODOLOGY

The MNIST open source database contains 70,000 images and labels of handwritten digits. Since training large neural networks can take a large amount of time, 24,000 of these images and labels were selected for this experiment. 20,000 of these images are used for training the network while the other 4,000 are used to test the performance of the network.

The algorithm in this paper uses a form of mini-batch gradient descent to train the network. Batch sizes that are too large can sacrifice the accuracy of the network while too small of batch sizes will force excessively long training times for the network. Considering these factors and after performing some initial tests, the 20,000 images were best broken up into 1,000 batches each of size 20.

To test the performance of the ProMoD backpropagation algorithm, the optimal coefficients for the proportional, momentum, and derivative term first have to be calibrated. This can be done by testing the accuracy of the network for various values of these coefficients

and selecting the ones which produce the highest accuracy. Once these coefficients are determined, the speed of the network can be compared against itself when different ProMoD terms are used in the algorithm. Three different configurations will be tested: 1) Base: P_k 2) Momentum: $P_k + I_k$ 3) ProMoD: $P_k + I_k + D_k$.

V. EXPERIMENTAL RESULTS

Figures 6-8 show the calibration graphs for the proportional, momentum, and derivative coefficients. The calibration graph in Figure 6 was configured with the base case and had a maximum accuracy of 89.62%. The calibration graph in Figure 7 was configured with the momentum case and had a maximum accuracy of 92.07%. The calibration graph in Figure 8 was configured with the ProMod case and had a maximum accuracy of 92.60%. Based on the calibration results, the accuracy of the network improved with the addition of the momentum and derivative terms. The addition of the momentum term had the most significant improvement. Table 2 shows the final calibration values for each method.

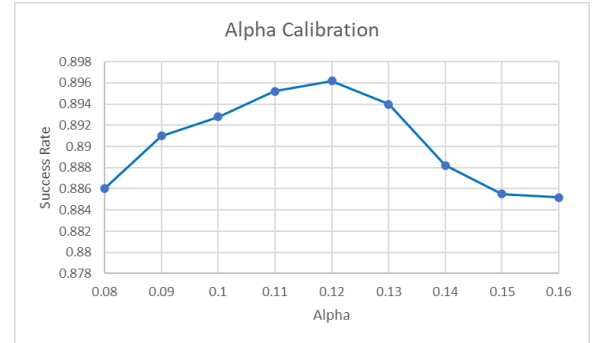


Figure 6. Proportional Coefficient Calibration

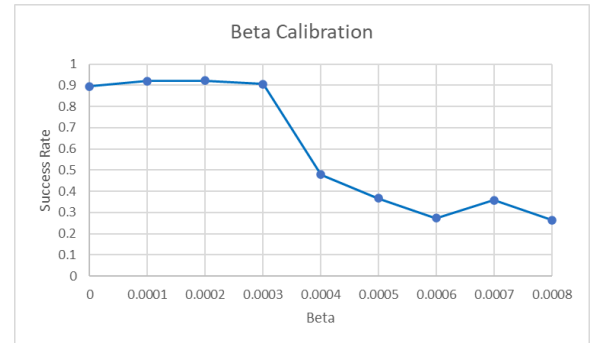


Figure 7. Momentum Coefficient Calibration

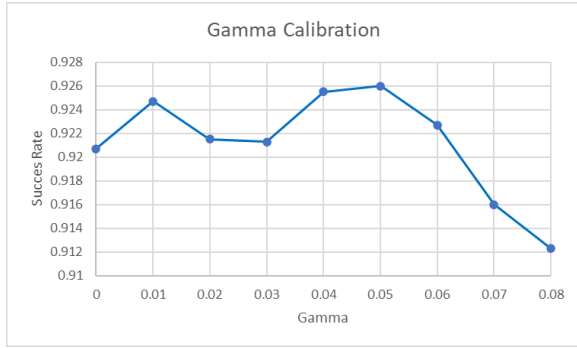


Figure 8. Derivative Coefficient Calibration

Table 2. Calibration Values for Each Method

	Base	Momentum	ProMoD
Proportional (α)	0.12	0.12	0.12
Momentum (β)	N/A	0.0001	0.0001
Derivative (γ)	N/A	N/A	0.05

Table 3 and Figure 9 show a tabular and visual comparison of the training speed for the base, momentum, and ProMoD methods. The momentum and ProMoD methods have similar accuracies and are better than the base method.

As far as comparing the speed goes, we use the 89.62% as a benchmark accuracy since this was the accuracy the base method was able to reach after 20,000 images. The momentum method reached a maximum accuracy of 92.37% and surpassed this benchmark with 13,000 training images. This means the momentum method was 1.43 times faster than the base method. The ProMoD method reached a maximum accuracy of 92.60% and surpassed the benchmark with 14,000 training images. This means the ProMoD method was 1.54 times faster than the base method.

Table 3. Training Speed Comparison for all Three Methods

Training Images	Base	Momentum	ProMoD
1000	0.4945	0.4993	0.457

2000	0.7448	0.7528	0.7063
3000	0.8147	0.8215	0.8173
4000	0.8570	0.8600	0.8465
5000	0.8640	0.8710	0.8728
6000	0.8695	0.8745	0.8735
7000	0.8605	0.8725	0.8870
8000	0.8800	0.8852	0.8768
9000	0.8770	0.8752	0.8845
10000	0.8920	0.8975	0.8972
11000	0.8850	0.8945	0.8870
12000	0.8898	0.8958	0.8925
13000	0.8898	0.8940	0.8988
14000	0.8735	0.9005	0.9040
15000	0.8735	0.8665	0.7927
16000	0.8910	0.9080	0.9207
17000	0.8890	0.9130	0.9207
18000	0.8835	0.9090	0.9197
19000	0.9040	0.9237	0.9193
20000	0.8962	0.9207	0.9260

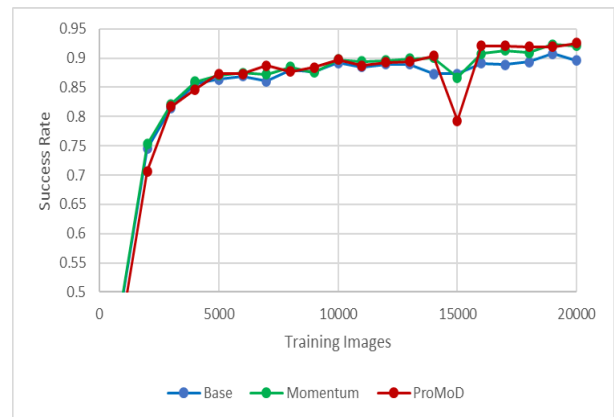


Figure 9. Visual Comparison of Training Speed
for all Three Methods

VI. CONCLUSION

The momentum and ProMoD implementations were undoubtedly superior in recognition accuracy and training speed to the base method. However, both of these methods had similar performance metrics when compared against one another. This seems to suggest that adding the derivative term to the backpropagation algorithm had little to no effect on the performance of the network.

This result however is inconclusive and is in stark contrast to [5] where adding the derivative term did have a strong effect on the performance. The CNN network in this paper was relatively small and does not feature state of the art accuracy. For the future, the CNN will have to be further developed until it has comparable performance to the network in [5] which has an accuracy near 96%. Perhaps also the CNN can be slightly modified to accommodate more general image processing applications. With more time and development, this project can have more conclusive results about the benefits of the ProMoD backpropagation as compared to the momentum backpropagation.

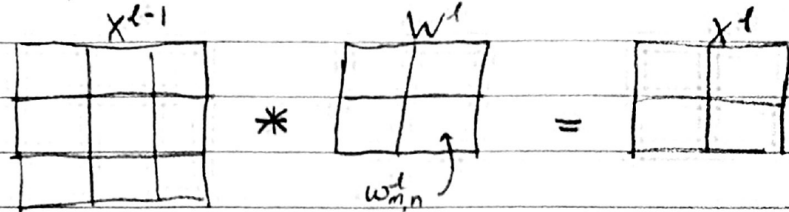
REFERENCES

- [1] U. R. Babu, Y. Venkateswarlu and A. K. Chintha, "Handwritten Digit Recognition Using K-Nearest Neighbor Classifier," 2014 World Congress on Computing and Communication Technologies, 2014, pp. 60-65, doi: 10.1109/WCCCT.2014.7.
- [2] G. Katiyar and S. Mehfuz, "SVM based off-line handwritten digit recognition," 2015 Annual IEEE India Conference (INDICON), 2015, pp. 1-5, doi: 10.1109/INDICON.2015.7443398.
- [3] T. Sasao, Y. Horikawa and Y. Iguchi, "Handwritten Digit Recognition Based on Classification Functions," 2020 IEEE 50th International Symposium on Multiple-Valued Logic (ISMVL), 2020, pp. 124-129, doi: 10.1109/ISMVL49045.2020.00-18.
- [4] C. Zhang, Z. Zhou and L. Lin, "Handwritten Digit Recognition Based on Convolutional Neural Network," 2020 Chinese Automation Congress (CAC), 2020, pp. 7384-7388, doi: 10.1109/CAC51589.2020.9326781.
- [5] A. Gürhanlı, "Accelerating convolutional neural network training using promod backpropagation algorithm," *IET Image Processing*, vol. 14, no. 13, pp. 2957-2964, 2020.
- [6] Y. Hou and H. Zhao, "Handwritten digit recognition based on depth neural network," 2017 International Conference on Intelligent Informatics and Biomedical Sciences (ICIIBMS), 2017, pp. 35-38, doi: 10.1109/ICIIBMS.2017.8279710.
- [7] S. Aly and S. Almotairi, "Deep Convolutional Self-Organizing Map Network for Robust Handwritten Digit Recognition," in *IEEE Access*, vol. 8, pp. 107035-107045, 2020, doi: 10.1109/ACCESS.2020.3000829.
- [8] J. Brownlee, "A gentle introduction to the rectified linear unit (ReLU)," *Machine Learning Mastery*, 20-Aug-2020. [Online]. Available: <https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/#:~:text=The%20rectified%20linear%20activation%20function,otherwise%2C%20it%20will%20output%20zero.> [Accessed: 10-May-2022].
- [9] Jefkine, "Backpropagation in Convolutional Neural Networks," *DeepGrid*, 05-Sep-2016. [Online]. Available: <https://www.jefkine.com/general/2016/09/05/backpropagation-in-convolutional-neural-networks/>. [Accessed: 09-May-2022].

APPENDIX

A. Equation (11) Derivation. Credits to [9] for the derivation.

Given an input feature map X^{l-1} , a weight kernel W^l , and bias b^l , we are looking to find the change to the cost function E as a result of a change to a single pixel $w_{m,n}^l$.



X^{l-1} has dimensions $H \times W$ and W has dimensions $K_1 \times K_2$.

$$\frac{\partial E}{\partial w_{m,n}^l} = \sum_{i=0}^{H-K_1} \sum_{j=0}^{W-K_2} \frac{\partial E}{\partial x_{i,j}^l} \frac{\partial x_{i,j}^l}{\partial w_{m,n}^l} = \sum_{i=0}^{H-K_1} \sum_{j=0}^{W-K_2} \delta_{i,j}^l \frac{\partial x_{i,j}^l}{\partial w_{m,n}^l}$$

$$\Rightarrow \frac{\partial x_{i,j}^l}{\partial w_{m,n}^l} = \frac{\partial}{\partial w_{m,n}^l} (w_{0,0}^l x_{i+0,j+0}^{l-1} + \dots + w_{m,n}^l x_{i+m,j+n}^{l-1} + \dots + b^l)$$

$$= \frac{\partial}{\partial w_{m,n}^l} (w_{m,n}^l \cdot x_{i+m,j+n}^{l-1})$$

$$= x_{i+m,j+n}^{l-1}$$

$$\Rightarrow \frac{\partial E}{\partial w_{m,n}^l} = \sum_{i=0}^{H-K_1} \sum_{j=0}^{W-K_2} \delta_{i,j}^l \cdot x_{i+m,j+n}^{l-1}$$

$$\frac{\partial E}{\partial w_{m,n}^l} = \text{rot}_{180} \{ \delta_{i,j}^l \} * x_{m,n}^{l-1}$$

B. MATLAB Code of Convolutional Neural Network

```
% Load training images from the MNIST database
trainIm = loadImages('MNIST/train-images.idx3-ubyte');
trainIm = reshape(trainIm, 28, 28, []);
```

```
% Load test images from the MNIST database
testIm = loadImages('MNIST/t10k-images.idx3-ubyte');
testIm = reshape(testIm, 28, 28, []);
```

```
% Load training labels from the MNIST database
trainLb = loadLabels('MNIST/train-labels.idx1-ubyte');
trainLb(trainLb == 0) = 10;
```



```

% Load test labels from the MNIST database
testLb = loadLabels('MNIST/t10k-labels.idx1-ubyte');
testLb(testLb == 0) = 10;

% Set the number of training and testing images
numtrain = 1000;
numtests = 4000;
trainIm = trainIm(:, :, 1:numtrain);
trainLb = trainLb(1:numtrain);
testIm = testIm(:, :, 1:numtests);
testLb = testLb(1:numtests);

% Randomize parameters of the CNN
rand('seed',1)
randn('seed',1)
C1 = 0.1*randn([9 9 4]);
C2 = 0.1*randn([5 5 8]);
F1 = 0.1*rand([10 49*size(C2,3)]);

% Train the Network
for epoch = 1:1

    % init derivative matrices
    dC1_derivative = zeros(size(C1));
    dC2_derivative = zeros(size(C2));
    dF1_derivative = zeros(size(F1));

    % init momentum matrices
    dC1_momentum = zeros(size(C1));
    dC2_momentum = zeros(size(C2));
    dF1_momentum = zeros(size(F1));

    % init previous matrices
    dC1_prev = zeros(size(C1));
    dC2_prev = zeros(size(C2));
    dF1_prev = zeros(size(F1));

    % divide training images into batches
    num_batches = 20;
    batches = 1:num_batches:(length(trainLb)-num_batches+1);

    % For each batch...
    for batch = 1:length(batches)

        % init gradient
        dC1 = zeros(size(C1));
        dC2 = zeros(size(C2));
        dF1 = zeros(size(F1));

        begin = batches(batch);

```

```

% For each training image in the batch...
for i = begin:begin+num_batches-1

    % Feed the image through the network
    input = trainIm(:, :, i);
    C1_out = convolve(input, C1);          % First convolution layer
    C1_out = max(0, C1_out);               % ReLU activation function
    S1_out = Pool(C1_out);                 % First pooling layer
    C2_out = convolve(S1_out, C2);          % Second convolution layer
    C2_out = max(0, C2_out);               % ReLU activation function
    S2_out = Pool(C2_out);                 % Second pooling layer
    S2_flat = reshape(S2_out, [], 1);      % Flatten to 1D array
    F1_out = F1 * S2_flat;                 % Connection layer
    output = exp(F1_out) / sum(exp(F1_out)); % Softmax activation function

    % Find the desired output from the corresponding training label
    desired = zeros(10, 1);
    desired(sub2ind(size(desired), trainLb(i), 1)) = 1;

    % Calculate the error and backpropagate it through the network
    error = desired - output;
    F1_update = error * S2_flat';
    S2_error = F1' * error;
    S2_error = reshape(S2_error, size(S2_out));
    C2_error = zeros(size(C2_out));

    for k = 1:size(C2,3)
        C2_error(:, :, k) = kron(S2_error(:, :, k), ones([2 2])) ./ 4;
    end

    C2_delta = (C2_out > 0) .* C2_error;
    C2_update = zeros(size(C2));

    pad = (size(C2,1) - 1) / 2;
    for k = 1:size(C2,3)
        for j = 1:size(S1_out,3)
            C2_update(:, :, k) = C2_update(:, :, k) + conv2(padarray(S1_out(:, :, j), [pad pad]),
rot90(C2_delta(:, :, k), 2), 'valid');
        end
    end

    C1_error = zeros(size(C1_out));

    for k = 1:size(C1,3)
        C1_error(:, :, k) = kron(C2_error(:, :, k), ones([2 2])) ./ 4;
    end

    C1_delta = (C1_out > 0) .* C1_error;
    C1_update = zeros(size(C1));

    pad = (size(C1,1) - 1) / 2;

```

```

for k = 1:size(C1,3)
    C1_update(:,:,k) = conv2(padarray(input(:,:,k), [pad pad]), rot90(C1_delta(:,:,k), 2), 'valid');
end

% Update each gradient
dC1 = dC1 + C1_update;
dC2 = dC2 + C2_update;
dF1 = dF1 + F1_update;

end

% ProMoD coefficients
alpha = 0.12;
beta = 0.0001;
gamma = 0.05;

% Average the gradients over each image in the batch
dC1 = dC1 / num_batches;
dC2 = dC2 / num_batches;
dF1 = dF1 / num_batches;

% Update the momentum matrices
dC1_momentum = dC1_momentum + dC1;
dC2_momentum = dC2_momentum + dC2;
dF1_momentum = dF1_momentum + dF1;

% Update the derivative matrices
dC1_derivative = dC1 - dC1_prev;
dC2_derivative = dC2 - dC2_prev;
dF1_derivative = dF1 - dF1_prev;

% Update the previous matrices
dC1_prev = dC1;
dC2_prev = dC2;
dF1_prev = dF1;

% Update Parameters
C1 = C1 + alpha * dC1 + beta * dC1_momentum + gamma * dC1_derivative;
C2 = C2 + alpha * dC2 + beta * dC2_momentum + gamma * dC2_derivative;
F1 = F1 + alpha * dF1 + beta * dF1_momentum + gamma * dF1_derivative;

end

end

% Test the Network
correct = 0;
for i = 1:length(testLb)

    % Feed the image through the network
    input = testIm(:,:,i);

```

```

C1_out = convolve(input, C1);      % First convolution layer
C1_out = max(0, C1_out);          % ReLU activation function
S1_out = Pool(C1_out);            % First pooling layer
C2_out = convolve(S1_out, C2);    % Second convolution layer
C2_out = max(0, C2_out);          % ReLU activation function
S2_out = Pool(C2_out);            % Second pooling layer
S2_flat = reshape(S2_out, [], 1); % Flatten to 1D array
F1_out = F1 * S2_flat;            % Connection layer
output = exp(F1_out) / sum(exp(F1_out)); % Softmax activation function

% Find the neuron with the strongest activation and see if it matches
% with the test label
[~,j] = max(output);
if j == testLb(i)
    correct = correct + 1;
end
end

% Display the results
fprintf('The accuracy of the network is %.4f\n', correct / length(testLb));

```