Overview Peter A Noble PhD

December 16, 2023 Email: panoble@gmail.com

Convolutional Neural Networks (CNNs) are commonly used for image classification tasks, and the MNIST dataset serves as a popular resource for practicing and comprehending these networks. The MNIST comprises 60,000 handwritten digits for training machine learning models and an additional 10,000 digits for testing the models (Figure 1). Introduced in 1998, the MNIST has since become a standard benchmark for various classification tasks. Download the MNIST PNG files to your Desktop from this site: https://github.com/DeepLenin/fashion-mnist_png/tree/master.

Here I present two coding projects: Pytorch CNN and C++ CNN. Pytorch was used to create the model in terms of architecture and generate weights and biases, and the C++ program uses the weights and biases to generate output, independent of the Pytorch model, so that it can be implemented elsewhere (e.g., internet).

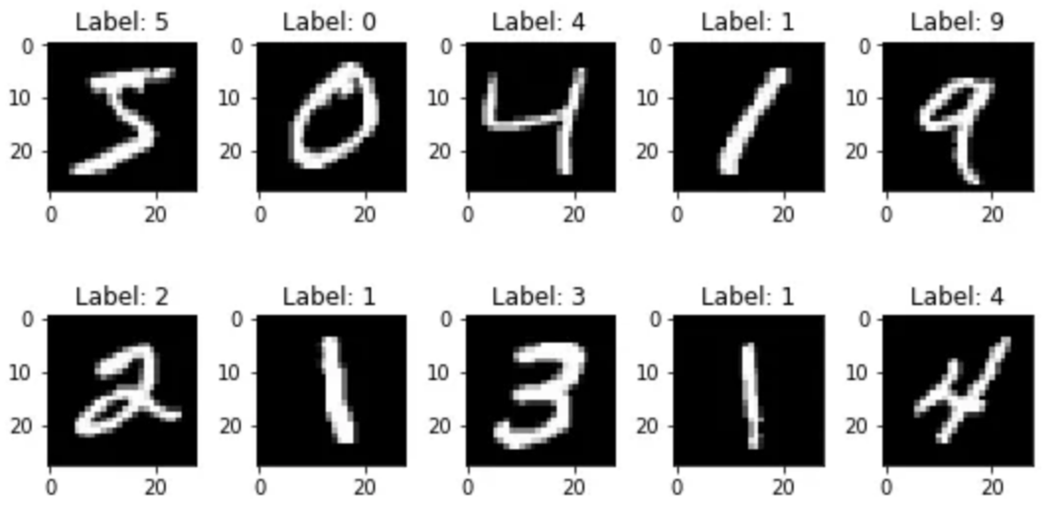


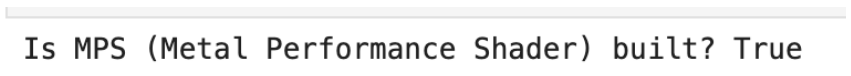
Figure 1. Examples of handwritten digits from the MNIST data set.

Part 1. Pytorch code

The PyTorch code from Jupyter Notebooks imports PNG images previously downloaded to your Desktop. The dataset is then split into 80% for training and 20% for testing. The images are processed in batches of 32 PNGs at a time. The testing images are saved for assessment analysis. The Convolutional Neural Network (CNN) is trained for 1 epoch in this example, with 64 filters (adjustable by the user). Once training is complete, the model's performance is evaluated using the testing dataset. Additionally, I've included a section for saving the weights and biases to the Desktop to develop applications outside of PyTorch.

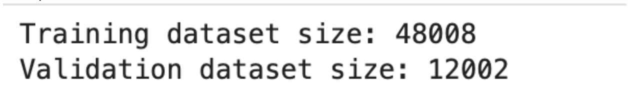
Pytorch Modeling Results

1. The program issues a statement on whether the Metal Performance Shader (MPS) was built successfully or not. MPS, a framework by Apple for iOS and MacOS, is designed to accelerate various image and signal processing tasks on GPUs. Leveraging parallel processing capabilities, developers can achieve high-performance computation in graphics, image processing, and machine learning applications.



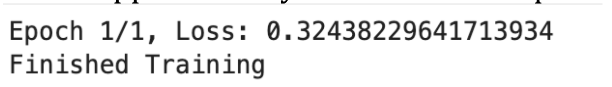
```
Is MPS (Metal Performance Shader) built? True
```

2. The training dataset consists of 48,008 files, and the validation dataset comprises 12,002 files.



```
Training dataset size: 48008
Validation dataset size: 12002
```

3. For demonstration purposes, the program is set to run 1 epoch. Users are encouraged to increase epochs for training (e.g., 10 epochs). The results indicate a loss of approximately 0.324 after one epoch.

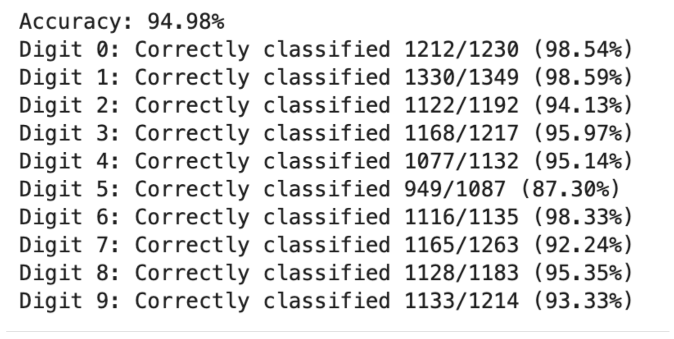


```
Epoch 1/1, Loss: 0.32438229641713934
Finished Training
```

4. The program saves the model, including architecture, weights, and biases, for later use.

5. After loading the validation dataset, the program calculates the accuracy for each digit. The overall accuracy for the model trained with just one epoch is 94.98%. Digit 5 exhibits the lowest accuracy, while digit 1 has the highest.

6. In the final part of the Pytorch program, the program saves the weights and biases in text format, facilitating future analyses. These can be used to build the model in C++ or MS Excel.



```
Accuracy: 94.98%
Digit 0: Correctly classified 1212/1230 (98.54%)
Digit 1: Correctly classified 1330/1349 (98.59%)
Digit 2: Correctly classified 1122/1192 (94.13%)
Digit 3: Correctly classified 1168/1217 (95.97%)
Digit 4: Correctly classified 1077/1132 (95.14%)
Digit 5: Correctly classified 949/1087 (87.30%)
Digit 6: Correctly classified 1116/1135 (98.33%)
Digit 7: Correctly classified 1165/1263 (92.24%)
Digit 8: Correctly classified 1128/1183 (95.35%)
Digit 9: Correctly classified 1133/1214 (93.33%)
```

Part 1. C++ code for modeling CNN

The C++ program called 'cnn.cpp' provides an understanding of how CNN Pytorch models the data. The Pytorch program uses binary images from the MNIST dataset, presumably because they allow faster processing than text images.

For the C++ program, we need to implement arrays of numbers including the input array (i.e., text representation of image), the weights and the biases, and the kernels, which are used to represent the weights of in the form of 2 by 2 arrays. The weights of the first layer (weights_0.txt) are fed into the kernel that moves across the image array and down, one pixel at a time.

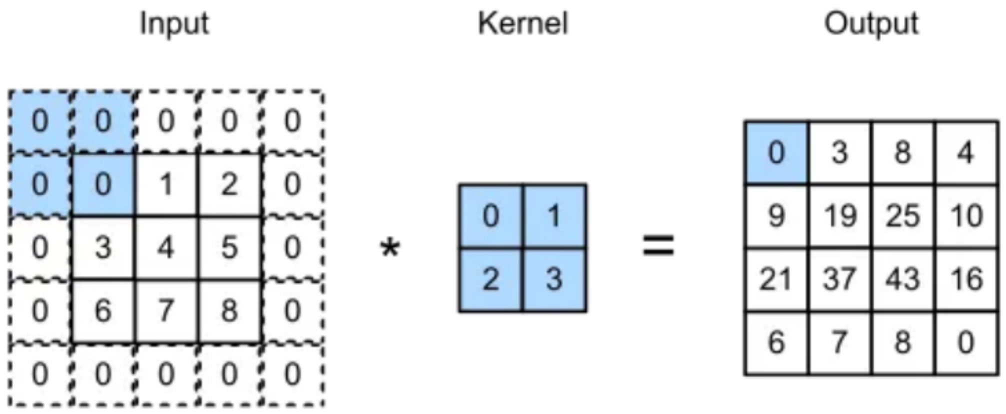


Fig 2. The size of the kernel is 2 × 2. (Image is downloaded from google.)

Here is how the convolution layer works. At each position, the corresponding pixel intensity is multiplied by the kernel value.

0 (input) x 0 (kernel weight) =0
0 (input) x 1 (kernel weight) =0
0 (input) x 2 (kernel weight) =0
0 (input) x 3 (kernel weight) =0

Therefore the output equals 0 + the first bias_0.txt. If the value is < 0, the value is set to 0 by the Rectified Linear Unit (ReLu) activation function. Then the kernel moves over one position and below the calculation.

0 (input) x 0 (kernel weight) =0
0 (input) x 1 (kernel weight) =0
0 (input) x 2 (kernel weight) =0
1 (input) x 3 (kernel weight) =3

Therefore the output equals 3 + the second bias_0.txt. If the value is < 0, the value is set to 0 by the ReLu activation function. Then the kernel moves across one position.

Converting PNG images to text arrays. ImageMagik (https://imagemagick.org/) was used to convert PNG files to PPM files. The conversion involved three steps. The first step is implemented as a Unix land command:

```
Convert file_name.png  -compress none file_name.ppm
```

The next steps involve implementing a C++ program to remove the first three lines of the PPM file that contains file information, specifically type (P3 or P6), width, height, and color type (BRGB).

The rest of the file is a pixel array of width x height with pixel intensities ranging from 0 to 255. There are three colors represented in the array, Red, Green and Blue. The C++ program is called 'image_look.cpp'. The final product is a text array with a specific width of 84 pixels and height of 28 pixels. In the example, you can convert the example PNG (77.png) (which represents the digit 2) to a text file called 77.txt.

The complete 3-step procedure is:

```
$ g++ image_look.cpp –o image_look              // compile the C++ program
$ convert 77.png  -compress none 77.ppm         // convert using ImageMagik
$ ./image_look 77.ppm 77.txt                    // convert ppm to text array
```

Pooling Next, the outputs are pooled to the maximum value of the outputs. The maximum value of the first 4 outputs (i.e., 0,3,9,19) is 19. The maximum of the second outputs (8,4,25,10) is 25. The maximum of the third outputs (21,37,6,7) is 37 and the maximum of the fourth output (43, 16,8, 0) is 43. Therefore the final pooled output is: 19, 25, 37 and 43. Each of these values will be multiplied by a second set of weights in the output layer (n=10 digits). The sum of the pooled output multiplied by the weight_1.txt for each digit is determined and the biases_0.txt are added.

The last step applies Softmax to the 10 outputs (one for each digit). The Softmax is a mathematical function that takes a vector of arbitrary real-valued scores and transforms them into a probability distribution.

$$\mathrm{Softmax}(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

The Softmax function essentially maps the input scores to a probability distribution, making it suitable for problems where the goal is to assign an input to one of multiple classes. The predicted class corresponds to the index with the highest probability.

Assessment of shuffling PNG images with noise on model performance.
A C++ program called 'add_shuffle.cpp' was devised to randomly shuffle pixel intensities in order to assess the effects of noise on the model accuracy. The models consisted of 64 filters that underwent 100 epochs. Figure 3 shows that model performance drastically declined with added noise.

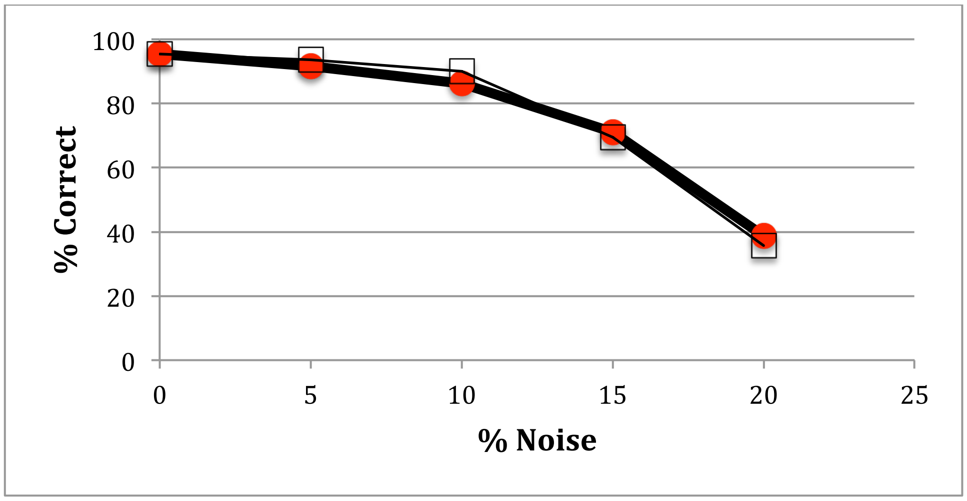


Figure 3. Effects of shuffling pixel intensities of model performance for two independent models.