CNN implementation with MNIST Dataset

Yetong Ding

August 13, 2022

1 Introduction

A Convolutional Neural Network (CNN) is generally implemented as a neural network for analysing visual imagery[5]. In this study, we implemented the CNNs on the MNIST (Modified National Institute of Standards and Technology) dataset[6] so that we are able to correctly identify digits from tens of thousands of handwritten images.

2 Literature

In this study, the MINIST dataset was used for training and evaluation, along with the PyTorch, TorchVision, ArgParse, and Time libraries.

2.1 MINIST

MNIST[6] is a large collection of handwritten digits used to train various image processing systems. The MNIST dataset consists of 60,000 training images and 10,000 test images, all of which are 28x28 pixel greyscale handwritten digits between 0 and 9, as shown in figure 1. Since its release in 1999, this classic set of handwriting images has been used to evaluate classification algorithms.

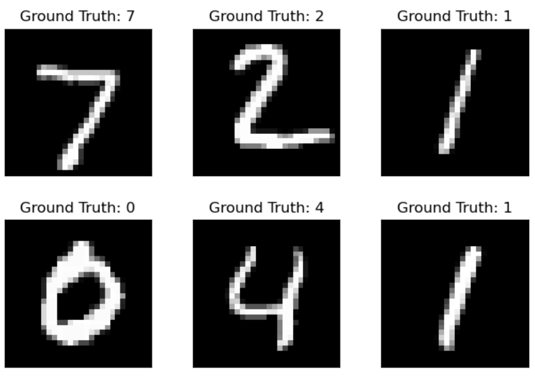

Figure 1: MNIST Dataset Sample with Ground Truth Labels

2.2 PyTorch

PyTorch[2] was developed primarily by Meta AI. It is an open source machine learning framework based on the Torch library that is used for applications such as computer vision and natural language processing. In our work, the PyTorch version is 1.10.1 with CUDA 11.3.

2.3 Torchvision

Torchvision[3] library is part of PyTorch framework. The torchvision package contains popular computer vision datasets, model architectures, and image transformations.

3 Methodology

3.1 Load Data

To improve the code readability and modularity, two PyTorch data primitives are used to decouple dataset code from model training code: *DataLoader* and *Data*. Datasets that permit the use of both pre-loaded datasets and one's own data. Dataset stores the samples and their respective labels, and DataLoader wraps an iterable around Dataset to facilitate access to the samples[1].

We transformed the MNIST dataset to Tensors of normalized range $[-1, 1]$, and we loaded with the following parameters[1][4]:

- ***root***: The path of stored training/test data
- ***train***: The specific training/test dataset
- ***download=True***: Downloads the data from the internet if it's not available at root
- ***transform***: The feature transformations
- ***target transform***: The label transformations

3.2 Define CNN Model

To begin, we defined the model with three convolutional layers, **nn.Conv2D()**, and one full-connected layer, **nn.Linear()**. For each CNN layer, we used batch normalisation, and a max-pooling layer. We also added the ReLU activation function, which helps the network learn non-linear decision boundaries[4]. Each output of the last layer is the input of the current layer. It is worthwhile to mention that the input channel is 1 due to the greyscale images. Furthermore, we feed features to the model using forward function.

3.3 Model Training and Testing

In this stage, we simply looped over our data iterator, fed the inputs to the network and trained with ADAM optimizer. Meanwhile, we used cross-entropy loss function for multi-class classification. After we trained the network for 10 passes over the training dataset, we predicted the class labels from the neural network against the ground-truth. Following that, the higher the prediction output, the more likely the network thinks the image of the predicted class is. As a result, we recorded the network predictions, and calculated the corresponding accuracy.

4 Results

4.1 Accuracy of Prediction

The overall accuracy of the network on the test image is approximately **98.6**%. Table 1 presents the accuracy on the test dataset by 10 digit classes. The accuracies of all classes are above 97%, and class 2 reaches the highest score of 99.8%.

Class Name	Accuracy (%)
0	99.5
1	**99.8**
2	97.3
3	98.5
4	99.4
5	97.5
6	98.0
7	98.7
8	99.3
8	98.7

Table 1: Accuracy on Test Data by Categories

4.2 Training Loss

In the training steps, we computed the training loss of each epoch by using the cross-entropy function and stored it in back propagation so that we could update the parameters during iterations. Figure 2 displays the plot of training loss over 10 times epochs. The training loss drops dramatically in the first epoch and then approaches zero slightly in subsequent epochs. The results demonstrate that the CNN model predicts MNIST datasets with high accuracy and efficiency.

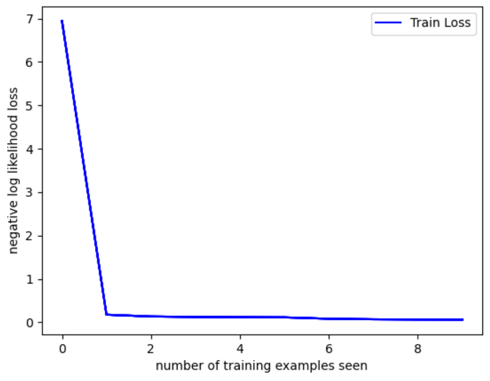

Figure 2: The Training Loss

5 Conclusions

In conclusion, we implemented a CNN to MNIST dataset. The accuracy of the neural network on test dataset is above 98%. To be more specific, Class digit 2 achieved the highest score, 99.8%. In addition, the training loss tends to approach zero, indicating that CNNs provide accurate predictions on the MNIST dataset.

References

[1] Datasets dataloaders — pytorch tutorials 1.11.0+cu102 documentation.

[2] Pytorch.

[3] torchvision — torchvision master documentation.

[4] Training a classifier — pytorch tutorials 1.5.0 documentation.

[5] Wikipedia Contributors. Convolutional neural network, 02 2019.

[6] Wikipedia Contributors. Mnist database, 02 2019.