University of Windsor

COMP 4740-1-R-2022W

ADV TOPICS: ARTIFICIAL INTELLIGENCE II

GUIDED BY: ROBIN GRAS

PROJECT REPORT SUBMISSION – 1

SUBMITTED BY:

Vidish Sharma – 110036779
Bhavya Bhimani – 110066194
Vaibhavi Lakhani – 110065860

PROBLEM STATEMENT:

Implement a simple CNN (Convolutional Neural Networks) architecture without using pre-existing complete network architecture or multi-layer blocks. Apply this model to the MNIST dataset and compare the results with existing CNN implementation.

About MNIST Database:

There are 70,000 instances in the MNIST database, 60,000 of which are for training and the rest are for testing. The database is made up of two diverse sources: NIST's Special Database 1 (which was gathered from high school students) and NIST's Special Database 3 (retrieved from Census Bureau employees). The training and test sets were chosen to avoid having the same writer work on both. More than 250 writers' samples are included in the training set [1].

IMPLEMENTATION:

We utilized the PyTorch framework to develop the solution and built a CNN model. We chose the PyTorch model over Keras and TensorFlow because it has greater training, evaluation, and debugging capabilities.

Preparing Data for training:
Data loaders were used to prepare data for training. Data Loader is an iterable that hides the complexity in a simple API. The Dataset retrieves the features and labels of our dataset one sample at a time. We pass samples in mini batches while training a model, reshuffle the data at each epoch to reduce model overfitting, and leverage Python's multiprocessing to speed up data retrieval.

Defining CNN:
Using torch.nn module, we developed a convolution neural network. All neural network modules will inherit from this base class. Two completely convolutional layers, the Relu activation function, and MaxPooling will be used.

Conv2D which applies a 2D convolution over an input signal made of several input planes.[2]
We have used the following parameters:
in_channels (int) - Number of channels in the input image
out_channels (int) - Number of channels produced by the convolution
kernel_size (int or tuple) - Size of the convolving kernel
stride (int or tuple, optional) - Stride of the convolution. Default: 1
padding (int or tuple, optional) - Zero-padding added to both sides of the input. Default: 0

We utilized the forward() pass to define how we compute our output using the layers and functions provided.

Loss Function:

We have used Cross Entropy Loss Function which computes loss between input and target.

Training Model:

Create a function called train () and pass number of epochs, model, and data loaders as input parameters. It uses clear gradients for training and computes gradients using back propagation. We have used over 100 epochs for training to get better accuracy.

num_epochs: Number of times our model will go through the entire training dataset.

Evaluating Model on Test Data:

Before executing inference, we must call model.eval() to set the dropout and batch normalization layers to evaluation mode.

model.train() informs the model that it is being trained. As a result, layers like dropout, batch norm, and others that behave differently on the train and in test procedures are aware of what is going on and may react accordingly.

To indicate that you are evaluating the model, use model.eval() or model.train(mode=False).

EXPERIMENTATION:

We have performed the testing on different platforms

- MacOS Monterey
    - CPU: Apple M1
    - RAM: 8 GB

  Accuracy obtained: 99%

- MacOS Mojave operating system
    - CPU: 1.6 GHz Intel Core i5
    - RAM: 8 GB 2133 MHz LPDDR
    - GPU: Intel UHD Graphics 617 1536 MB graphics

  Accuracy obtained: 98%

- Windows 10 operating system
    - CPU: Intel Core i5
    - RAM: 24 GB
    - GPU: Nvidia 980TF-4GB

  Accuracy obtained: 97.5%

COMPARISONS:

We referred to different existing CNN models. The results from using the CNN model of [4] states that CNN gave better results on 25% of MNIST and had the best baseline to train. This achieves only 98.32% of accuracy on digits. This CNN model has three convolutional layers in which each layer has 5x5 kernels and stride of one. A ten-class SoftMax with cross-entropy loss follows. The standard model, however, has two flaws. To begin, creating an effective CNN model necessitates a significant amount of training data. Second, CNN loses some information at the pooling layer, resulting in a lack of understanding of interrelationships between distinct components.

Secondly, when we used the model of [5], only 68.57 % of accuracy can be achieved on a random image dataset. They designed the CNN model with 4 convolutional layers with 32 filters in the first two and 64 filters in the next two. After two convolutional layers with strides of two, there were two max-pooling layers, each of which reduced the image. After each convolutional network, the activation strategy employed is Relu. After a pooling layer, there was a 0.25 dropout. The resultant Matrix was flattened and provided as the input to the fully connected network after the features were removed.

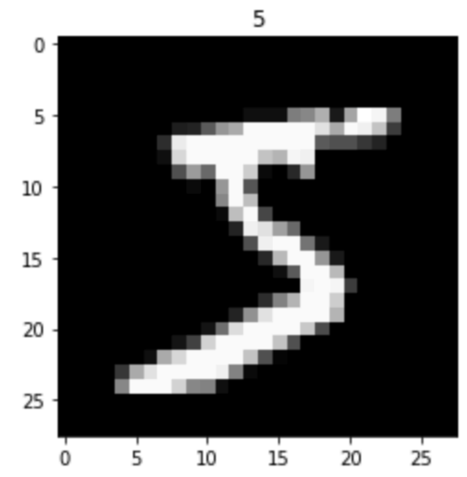
The random 300 photos supplied to the model with 200 epochs have a testing accuracy of 68.57 per cent. Upon further examination of the data, it was discovered that the model had difficulty detecting printed letters with sharp edges. With smooth-edged values, the model performed better.

Finally, we used the [6] model, which has a 97 percent accuracy average. They utilized a basic CNN model to obtain up to four separate feature maps from the first layer. The max pooling layer is used to compute the local average of the feature maps. There were two more layers, like the preceding two. It was discovered that the precision of specific digits, such as 0,1,7, was higher.

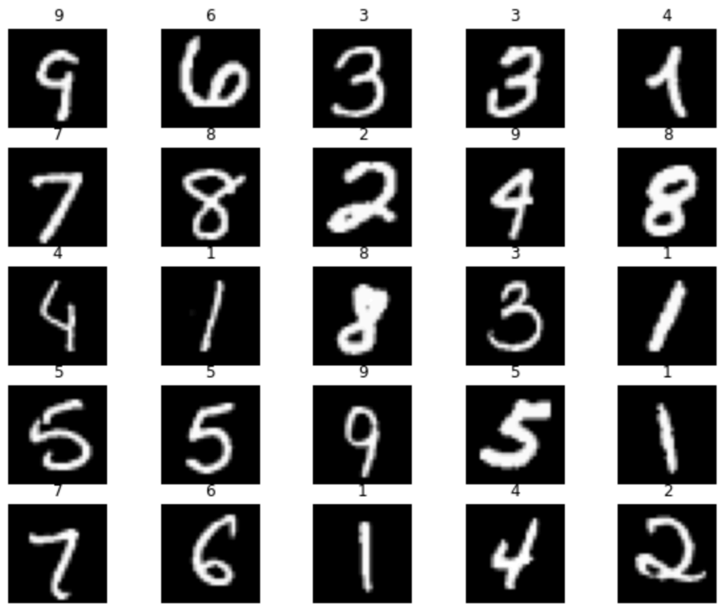
To sum up, we observed many CNN models and tracked their accuracy and performance. As a result, as described in the Implementation section, we developed our own CNN model with a 99 % accuracy.

RESULTS:

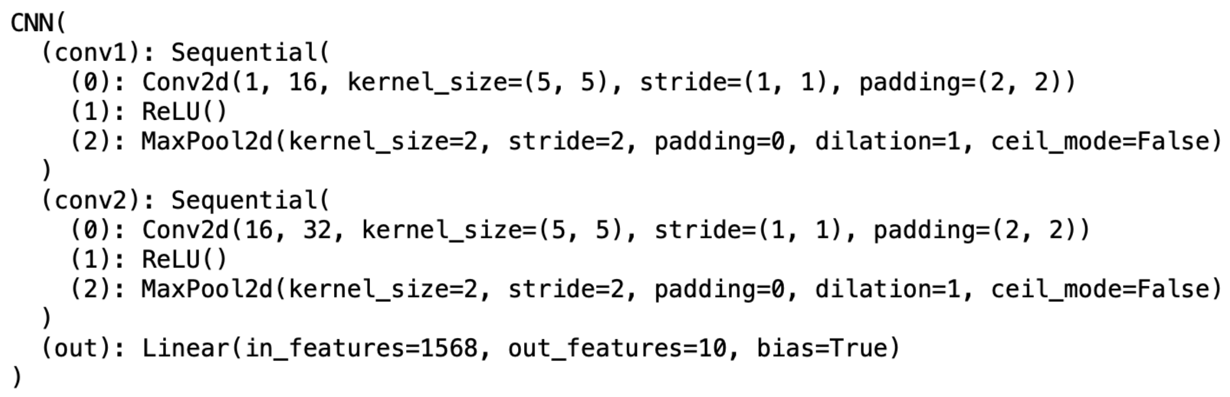
Below are the results of the output when we trained and tested our CNN model for 100 epochs.



Sample image in the MNIST testing dataset.

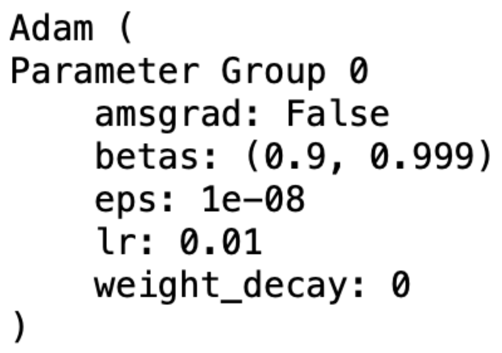


Sample dataset of the images and their labels for Training.



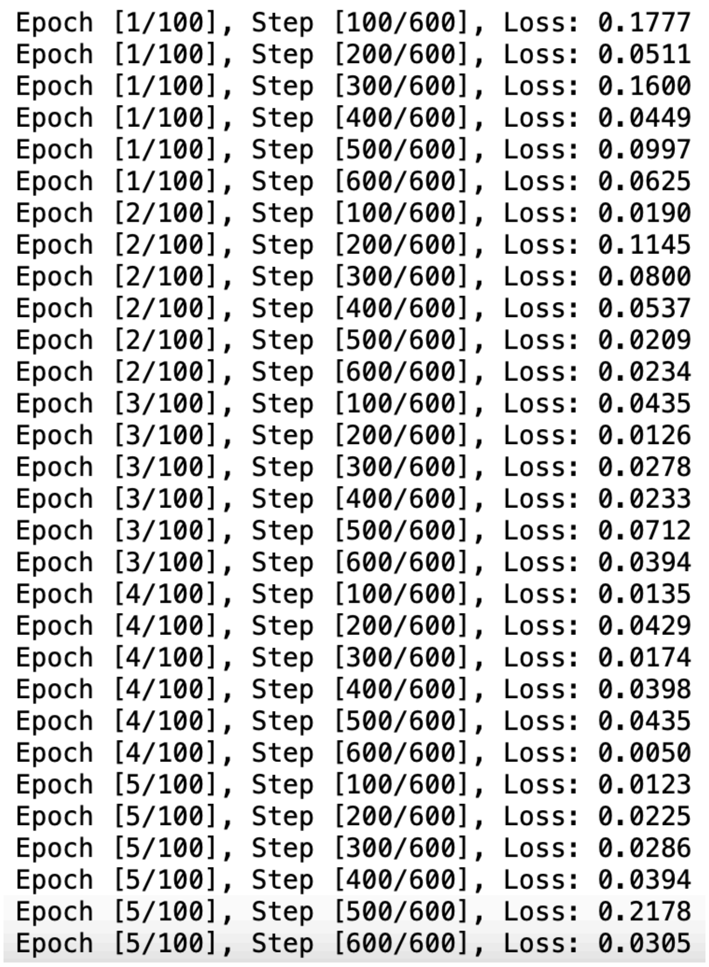
```
CNN(
  (conv1): Sequential(
    (0): Conv2d(1, 16, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (conv2): Sequential(
    (0): Conv2d(16, 32, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (out): Linear(in_features=1568, out_features=10, bias=True)
)
```

The Convolutional Neural Network Model



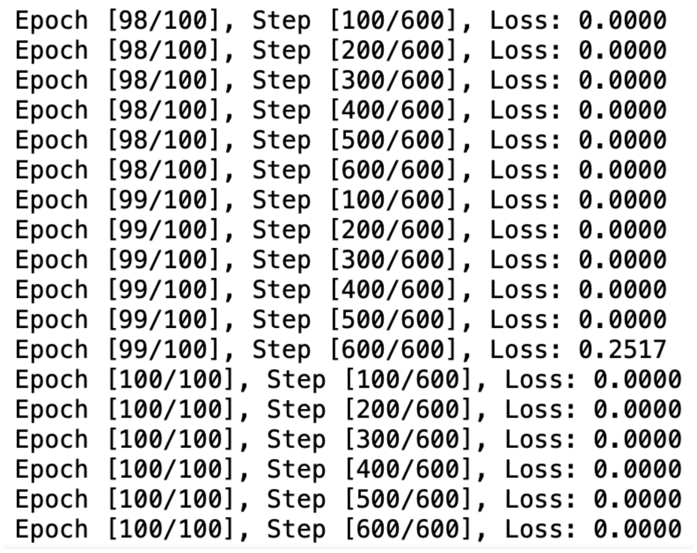
```
Adam (
Parameter Group 0
    amsgrad: False
    betas: (0.9, 0.999)
    eps: 1e-08
    lr: 0.01
    weight_decay: 0
)
```

The Optimizer in our CNN model



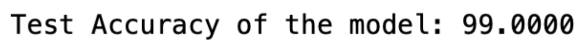
```
Epoch [1/100], Step [100/600], Loss: 0.1777
Epoch [1/100], Step [200/600], Loss: 0.0511
Epoch [1/100], Step [300/600], Loss: 0.1600
Epoch [1/100], Step [400/600], Loss: 0.0449
Epoch [1/100], Step [500/600], Loss: 0.0997
Epoch [1/100], Step [600/600], Loss: 0.0625
Epoch [2/100], Step [100/600], Loss: 0.0190
Epoch [2/100], Step [200/600], Loss: 0.1145
Epoch [2/100], Step [300/600], Loss: 0.0800
Epoch [2/100], Step [400/600], Loss: 0.0537
Epoch [2/100], Step [500/600], Loss: 0.0209
Epoch [2/100], Step [600/600], Loss: 0.0234
Epoch [3/100], Step [100/600], Loss: 0.0435
Epoch [3/100], Step [200/600], Loss: 0.0126
Epoch [3/100], Step [300/600], Loss: 0.0278
Epoch [3/100], Step [400/600], Loss: 0.0233
Epoch [3/100], Step [500/600], Loss: 0.0712
Epoch [3/100], Step [600/600], Loss: 0.0394
Epoch [4/100], Step [100/600], Loss: 0.0135
Epoch [4/100], Step [200/600], Loss: 0.0429
Epoch [4/100], Step [300/600], Loss: 0.0174
Epoch [4/100], Step [400/600], Loss: 0.0398
Epoch [4/100], Step [500/600], Loss: 0.0435
Epoch [4/100], Step [600/600], Loss: 0.0050
Epoch [5/100], Step [100/600], Loss: 0.0123
Epoch [5/100], Step [200/600], Loss: 0.0225
Epoch [5/100], Step [300/600], Loss: 0.0286
Epoch [5/100], Step [400/600], Loss: 0.0394
Epoch [5/100], Step [500/600], Loss: 0.2178
Epoch [5/100], Step [600/600], Loss: 0.0305
```

Training the model – Epochs 1 to 5 with the Loss of every step.



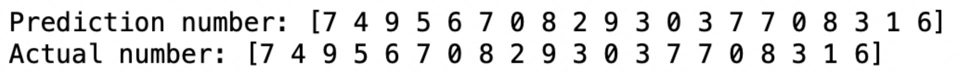
```
Epoch [98/100], Step [100/600], Loss: 0.0000
Epoch [98/100], Step [200/600], Loss: 0.0000
Epoch [98/100], Step [300/600], Loss: 0.0000
Epoch [98/100], Step [400/600], Loss: 0.0000
Epoch [98/100], Step [500/600], Loss: 0.0000
Epoch [98/100], Step [600/600], Loss: 0.0000
Epoch [99/100], Step [100/600], Loss: 0.0000
Epoch [99/100], Step [200/600], Loss: 0.0000
Epoch [99/100], Step [300/600], Loss: 0.0000
Epoch [99/100], Step [400/600], Loss: 0.0000
Epoch [99/100], Step [500/600], Loss: 0.0000
Epoch [99/100], Step [600/600], Loss: 0.2517
Epoch [100/100], Step [100/600], Loss: 0.0000
Epoch [100/100], Step [200/600], Loss: 0.0000
Epoch [100/100], Step [300/600], Loss: 0.0000
Epoch [100/100], Step [400/600], Loss: 0.0000
Epoch [100/100], Step [500/600], Loss: 0.0000
Epoch [100/100], Step [600/600], Loss: 0.0000
```

Training the model – Epochs 98 to 100 with the Loss of every step.



```
Test Accuracy of the model: 99.0000
```

Test Accuracy of our CNN model.



```
Prediction number: [7 4 9 5 6 7 0 8 2 9 3 0 3 7 7 0 8 3 1 6]
Actual number: [7 4 9 5 6 7 0 8 2 9 3 0 3 7 7 0 8 3 1 6]
```

The predicted numbers by our CNN model and the actual numbers.

REFERENCES:

[1] Alejandro B, Yago S, Pedro I. “A Survey of Handwritten Character Recognition with MNIST and EMNIST,” Applied Sciences, MDPI, 2019.

[2] [Online Resource] https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html

[3] [Online Resource] https://www.simplilearn.com/keras-vs-tensorflow-vs-pytorch-article

[4] Feiyang C, Nan C, Hanyang M, Hanlin H. “Assessing Four Neural Networks on Handwritten Digit Recognition Dataset (MNIST)”, Chuangxinban Journal of Computing, June 2018.

[5] Adhesh G, Diwanshi G, Sanjay S, Parimi S. “Validation of Random Dataset using an efficient CNN model trained on MNIST handwritten Dataset”, 2019 IEEE, 6th International Conference on Signal Processing and Integrated Networks (SPIN), 2019, pp. 602-606.

[6] Mahmoud G, Ashraf M. “A Comparative Study on Handwriting Digit Recognition Using Neural Networks,” 2017 IEEE International Conference on Promising Electronic Technologies, 2017, pp. 77-81.

[7] [Online Resource] https://medium.com/@nutanbhogendrasharma/pytorch-convolutional-neural-network-with-mnist-dataset-4e8a4265e118