Deep Learning Final Report: Convolutional Neural Networks

Nguyen Thanh Binh - 2440052
Information and Communication Technology
University of Science and Technology of Hanoi
Hanoi, Vietnam
binhnt2440052@usth.edu.vn

***Abstract*—In this project, I will try to make a Convolutional Neural Network without using any webiste packages, but internal packages such as `math` and internal packages, for easy to read. We will do the convolution steps, max pooling with strides, and flatten the input for Neural Network forward pass with ReLU activation function and Softmax function**

***Index Terms*—Deep learning, Convolutional Neural Network, forward pass.**

I. Introduction

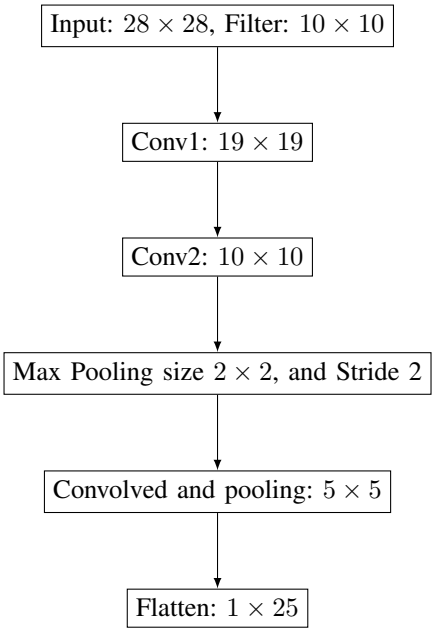
We all know the convolutional neural network can be used from developed websites for free such as TensorFlow, Keras, and PyTorch, however not us know much about how the package contents like, we just use it blindly. So, for today, we will try to do it without them. In this report, we will only cover the parts that has been done, except the backward pass, which is the most difficult part and took the longest, and it is not finished yet.

II. The dataset

We will use the MNIST dataset or hand-written digit number dataset for the model since it is from scratch so it may perform poorly and slowly. The MNIST dataset contains of 70000 images, each labeled with 10 classes (0 to 9). For the image size for the model, we will use the size of 28×28.

III. The architecture:

For the architecture of the convolutional neural network, we will have two convolutional layers, one max pooling layer and one flattened layer, which is also a fully connected layer. The input layer will have a size of 28×28, and the filter size we will use is 10×10, since we do two convolutions, the first convolution layer will have the size of 19×19, the next convolution layer will have the size of 10×10. All convolution layers are done with the default stride 1, which the filter take 1 step for sliding through the image. After finishing convolving, we will use the max pooling with the pooling size of 2×2, and with the stride of 2, to reduce the dimension for quicker computing. The final image size is reduced to 5×5. The following figure is demonstrated below:



IV. The model

A. Training model:

We will use 60000 image from the MNIST Dataset for training and 10000 for testing. We will do it in 5 epochs. However, the training time still took too long that we did not have a chance to get the results and perform the prediction. During training, the time it took for computing the convolution image is very complicated, as there are too many nested loops when perform in each layer. Then, we also have to do the linear forward pass, which requires the matrix multiplication. After convolving, weights and biases' values are randomly assigned. During the first epoch, since it is random, the loss is very high. And so, we need to do backprop to get not only the best weight, but also, perfect filter for this classification.

B. Backpropagation:

This is the most difficult task to perform since it is a very long way back for the model, not to mention of the risks of having the vanishing gradient, making the next epochs become worthless. The idea is to calculate the loss function, from the loss function, we calculate the gradient of each neuron

of the flatten layer. Once the gradient of the neuron layer is computed, we also compute the gradient of the weights and biases. To update weights and biases, we will subtract it:

$$w_j = w_j \times (1 - lr \times \delta_j)$$

Where lr is learning rate, δ_j is the gradient of the corresponding weight j. The same applies to bias.

For max pooling layer,, we will reshape the flattened gradient into a matrix matching the pooing output shape. Then, we pass the gradients back to the ReLU layer only at the positions where the max value occurred in the forward pass.

For each 2×2 pool, we will assign the gradient to the max corresponding value position.

Then, we will compute the gradient of the ReLU: 1 if positive, 0 otherwise. After the gradient is finished, we go back to the filter part, where we will update it by subtracting it to learning rate × gradient from the ReLU function.

This back and forth process will repeat 4 more times before we get to the most optimal filter and weights and biases.

C. Matrix Operations:

To do the basic operations, numerous nested loop algorithms must be written, it is included along with the main python file.

Assume that we have a list of list of float, to do multiplication, we must loop up to two times for the dot product of each vector in the lists.

For convolution, we have to do four nested loops in order to get the convolved layer, first two for the sliding, last two for the convolution sum.

For max pooling, we will take the multiple same positional elements from 2 lists of the list. And we will use the additional operation on list for appending and finding the max value, acting like a real max pooling in paper.

Flatten is simple, 2 nested loops for each float element and append it for making a list that still preserve the total entries. No reshaping function needed.

To perform the Linear function, we need to do the addition function as the requirement is: Both matrices or vectors must have identical sizes. Then it is simple, similar positional entries add up to each other.

D. Randomizing

We use the basic random genrator, the Linear Congruetual Generator.

V. Expected results and prediction on finish:

Because the final result is not done yet, due to the hardship we are occurring, we will have the final result once the model is fully completed. But as of now we can only give you the expected result. For the MNIST dataset, the performance is pretty long to finish.

The metric we will use to evaluate the model is the loss function, the lower the loss, the better the model:

This result was done in only 35 epochs from an already built model. But if we do it from scratch, it will take us more time to adjust, to fix carefully the backprop function. As of now, we have successfully done the convolution layers, max pooling and flatten.

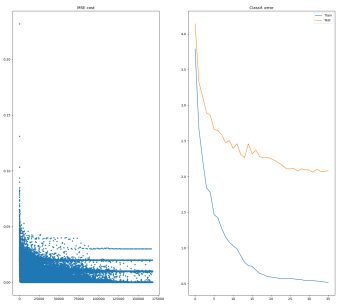


Fig. 1. The expected loss for the model

At first, we expect the first tried finished model will have a very poor perfomance, with high loss, and the flatten layer will have a very large float value, making the model fail to comply to our target. But as it is still on progress, we will try to address the problem and try to imporve it.

References