BITS F464 - Machine Learning
I Semester 2022-2023
Assignment : 1
Convolutional Neural Networks

Professor : Dr. Navneet Goyal
Group 2 Members :

- Ujjwal Jain (2019B4A70647P)
- Shobhit Jain (2019B3A70385P)

ACKNOWLEDGEMENT

We express gratitude to Dr. Navneet Goyal for his immense effort in guiding us and providing us the required resources and knowledge.

My sincerest gratitude to Birla Institute of Technology and Science, Pilani for providing us with such a great learning opportunity.

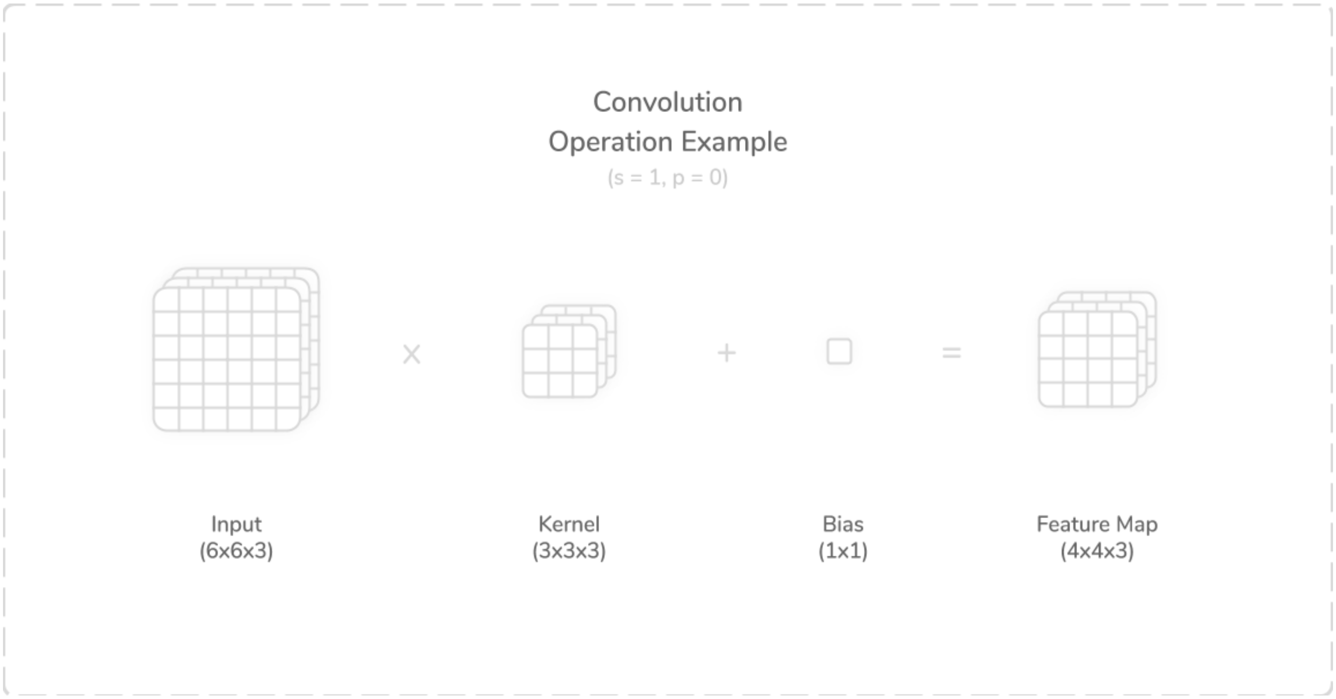
<u>Introduction to Convolutional Neural Networks</u>

Convolutional Neural Networks (CNNs) are a type of deep learning model that are commonly used for image recognition and classification tasks. They are composed of multiple layers, including convolutional layers, pooling layers, and fully connected layers.

CNNs are called "convolutional" because they use a mathematical operation called convolution to process the input data. In the case of image data, this operation involves sliding a small matrix of numbers (called a kernel or filter) over the input image and computing the dot product at each location. These operations are known as strides where the kernel is moved over the input in steps. The step size is variable. A larger step means that the kernel skips more space per step. This in turn leads to a lesser number of convolutions and hence a smaller output.For each given step are input and kernel are multiplied with weights and all are summed to get the result. Hence a feature map is produced containing the results of the convolutions. These maps may be an input to next layers and hence are passed through activation filters. The final feature map size is :

$[(\text{input Size} - \text{kernel Size} + 2 \times \text{padding}) / \text{stride}] + 1.$

This process allows the model to extract features from the input image, such as edges, corners, and textures.



The above figure shows the example of convolution operation 3 channel input of size 6x6, a 3 channel 3x3 kernel and a 1x1 bias. The convolution was implemented with no padding and one stride.This means that the kernel (3x3) will slide over input (6x6) from left to right hence generating the result (4x4) convolved feature map.

The convolutional layers in a CNN are followed by pooling layers, which downsample the output of the convolutional layers, reducing the dimensionality of the data and reducing the computational complexity of the model.

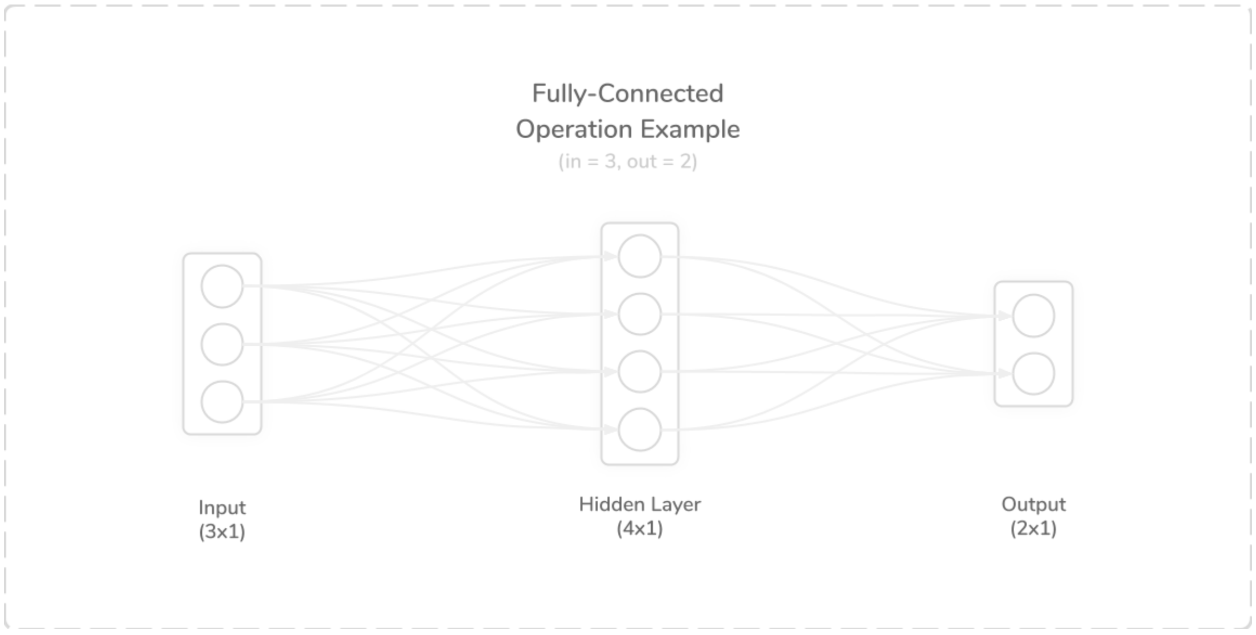
Two-dimensional subsampling layers down-sample input feature using windows or non-trainable kernels.The feature size is significantly reduced and helped to remove a network’s position reliance. The two conventional types of subsampling are max pooling and average pooling.The resulting feature for both contains either the average or maximum of the values present in each of the kernel.These layers can be implemented to include trainable parameters to aid with overall model learning.

This is important because it allows the model to learn more abstract features from the data, such as the shape of an object or the overall scene.

After the convolutional and pooling layers, the output is typically passed through one or more fully connected layers, which use the extracted features to make predictions.

Their operation generally involves multiplying the input by trainable weight vectors,along with a trainable bias which in turn can be summed to get the final result. The output is typically passed through activation functions, similarly to convolution layers.

For example, in the case of image classification, the fully connected layers would use the extracted features to classify the input image into one of a predefined set of classes.



The above image shows the example of a fully connected operation that has a 3x1 input, a 4x1 hidden layer and 2x1 output.

Overall, CNNs are a powerful tool for image recognition and classification tasks, and have achieved state-of-the-art performance on a variety of benchmarks. They are widely used in applications such as object detection, image segmentation, and facial recognition.

Implementation of Convolutional Neural Networks

The Convolutional Neural Network (CNN) code is implemented in C++ programming language.

The dataset used for the same is MNIST Dataset: Digit Recognizer. MNIST ("Modified National Institute of Standards and Technology") is the de facto “Hello World” dataset of computer vision. Since its release in 1999, this classic dataset of handwritten images has served as the basis for benchmarking classification algorithms. As new machine learning techniques emerge, MNIST remains a reliable resource for researchers and learners alike.

It consists of a training set of 60,000 grayscale images of handwritten digits (0-9), and a test set of 10,000 images.

The images in the MNIST dataset are 28x28 pixels in size, and each pixel is represented by a grayscale intensity value between 0 and 255. The dataset also includes the corresponding labels for each image, indicating the correct digit that is written in the image.

We aim to correctly identify digits from a dataset of tens of thousands of handwritten images.

1. The “Core.h” file includes :
    1.1. All the necessary header files
    1.2. Type declarations :
        1.2.1. vector<vector< double >> = layer
        1.2.2. vector<vector<vector<double>>> = matrix
    1.3. Helper functions :
        1.3.1. Cross_correlate : Two 2D matrices are passed as input and the cross_correlation operation is performed returning the resulting output matrix
        1.3.2. Cross_correlate3d : Two 3D matrices are passed as input and the cross_correlation operation is performed returning the resulting output matrix
        1.3.3. Rotatematrix : Returns the 180° rotation of the input matrix
        1.3.4. Convolution_full : Two 3D matrices are passed as input and the Full convolution (with padded matrix) operation is performed returning the resulting output matrix
        1.3.5. Dot_product : Two 2D matrices are passed as input and the Dot product operation is performed returning the resulting output matrix
        1.3.6. Matrix_multiplication : Two 2D matrices are passed as input and the matrix multiplication operation is performed returning the resulting output matrix
        1.3.7. Transpose : Returns the Transpose of a matrix

    1.4. Classes :
        1.4.1. Convolutional :

        1.4.1.1. Constructor takes in input shape(height, width, depth), number of kernels, kernel dimensions (considered square), boolean padding
        1.4.1.2. Initialize Kernel and Biases with random values
        1.4.1.3. Update kernel with learning rate and kernel gradient
        1.4.1.4. Update biases with the learning rate and output gradient
    1.4.2. Activation Layers :
        1.4.2.1. ReLU
        1.4.2.2. Sigmoid
        1.4.2.3. Tanh
        1.4.2.4. Softmax
    1.4.3. Reshape : Reshapes the output of the previous layer to be compatible with the input parameter of the dense layer (1D vector) and vice versa for back propagation
    1.4.4. Dense : Each dense layer acts as a hidden layer for the neural network
        1.4.4.1. Constructor with input dimensions and output dimensions
        1.4.4.2. Initialize Weights and Biases with random values
        1.4.4.3. Update Weights with the learning rate and weights gradient
        1.4.4.4. Update Bias with the learning rate and output gradient

All classes have their own forward and backward propagation methods

1.5. Loss Functions and Derivatives : Calculates the Error between the actual labels and predicted labels using the concept of Loss

2. The CNN.cpp file includes :
   - 2.1. The structure of the CNN , various layers are initialized globally
   - 2.2. Data Loader function to load the required data
   - 2.3. One Iteration function constitutes cycle of forward and backward with corresponding loss calculation returning the loss value
   - 2.4. Main function run the CNN for 5 epochs for images in the image_set output the loss values

Results and Conclusion

Given that it was a fully ground up implementation, the results weren't particularly encouraging. Particularly in terms of due to the model's optimization and the fact that logarithmic values frequently have NaN values, numerical precision employing Stochastic Gradient Descent, making it <u>quite susceptible to becoming stuck near local minima</u>. Time and computational resource limitations prevented us from making too many many explorations. Trial accuracy on the MNIST dataset, which was trained over around 15000 cases, was 50% and 30% testing accuracy.