Use of linear and matrix algebra in Convolutional neural networks

Aakash Gireesh Tripathi and Leela Pavan Kumar Parvathaneni
Linear and Matrix Algebra
11/28/2022

***Abstract*—In this report, we will discuss the use of linear and matrix algebra in a class of deep neural networks called Convolutional Neural Networks (CNNs). We also demonstrate the advantages of using CNNs in the task of classifying handwritten digits and compare its performance with a basic fully connected neural network (FCNN). Our implementation and results are openly available on GitHub.**[1]

I. Introduction

Artificial neural networks (ANNs), often called deep fully connected networks, are a class of machine learning algorithms that are inspired by the structure and function of biological neural networks. They are composed of interconnected units called neurons, which can process information. The signal at a neuron is a function of the sum of signals from all the neurons connected to it. The output of a neuron is determined by a non-linear activation function. The network is trained by adjusting the weights of the connections between neurons.

ANNs are used for a variety of tasks such as classification, regression, and clustering. They are used in applications such as speech recognition, image recognition, and natural language processing. But their performance is the most efficient when paired with a feature extraction algorithm [5].

A Convolutional Neural Network (CNN) is a class of deep neural networks most commonly used for image classification and recognition tasks. CNNs are composed of one or more convolutional layers and one or more fully connected layers. The convolutional layers are responsible for extracting features from the input image which is then fed as an input to a deep fully connected network which is responsible for classifying the input image [1].

II. Background

This section outlines the necessary background knowledge needed for the convolutional neural network.

[1]https://github.com/Aakash-Tripathi/CNN

A. What is convolution?

The kernel is a small matrix that is applied to the input image to extract features from the image. The kernel is also called a filter or a feature detector. The dimensions of the kernel can be, for example, 3x3. An example of a convolution operation can be seen in figure 1. It shows a 3x3 kernel being applied to a 4x4 input image. The kernel is applied to the input image by multiplying the kernel with the input image element by element. The result of the multiplication is then summed up to produce a single value. This process is repeated for all the elements in the input image. The result of the convolution operation is a feature map [4].

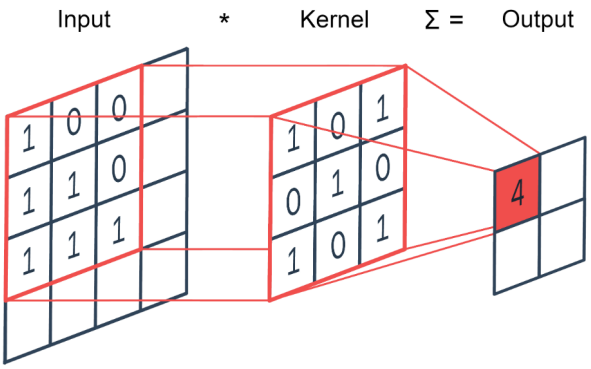

Fig. 1: Example of the convolution function

A convolution operation on the MNIST [2] dataset can be seen in figure 2 where the kernel is applied to the an image in the dataset. The result of the convolution operation is a feature map which is then used as an input to the fully connected network. The kernel is a learned parameter of the network and is updated during the training process.

Additionally sub-sampling is also used in CNN's. Subsampling is a process where the size of the feature map is reduced by pooling the values in the feature map.

We implement a max pooling layer in our CNN. Max pooling can be define by the following equation:

$$P_{i,j} = max_{k,l}(F_{i+k,j+l}) \tag{1}$$

where $P_{i,j}$ is the output of the pooling layer, $F_{i,j}$ is the input feature map, and k and l are the kernel size.

Kernel convolutions are not only used in CNN's, but are also a key element of many other computer vision algorithms. It is a process where features can be extracted from an image.

B. Neural Networks

A typical neural network is composed of an input layer, one or more hidden layers, and an output layer. The input layer is responsible for receiving the input data. The hidden layers are responsible for processing the input data and extracting features from it. The output layer is responsible for producing the output [5]. Figure 3 shows a typical neural network.

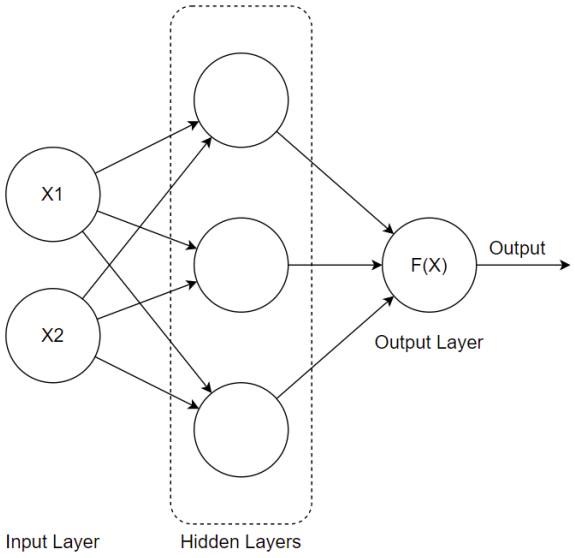

Fig. 3: A typical neural network

The input data is fed to the input layer. The input layer then passes the data to the first hidden layer. The first hidden layer then passes the data to the second hidden layer and so on. The output layer then produces the output.

The hidden layers are composed of neurons. Each neuron is connected to all the neurons in the previous layer. The output of a neuron is a function of the sum of the inputs from all the neurons in the previous layer. The output of a neuron is determined by a non-linear activation function. The weights of the connections between neurons are adjusted during the training process.

III. Convolutional Neural Networks

Training a neural network involves adjusting the weights of the connections between neurons and the training process can be summarized into two processes: forward propagation and backpropagation. In forward propagation, the input data is passed through the neural network. The output of the neural network is compared with the target output to calculate the error. In backpropagation, the error is used to adjust the weights of the connection between neurons. This process is repeated until the error is minimized [5].

A. Forward Propagation

The output of the convolution layers is a 2D matrix. This matrix is then flattened into a 1D vector and passed to the fully connected layers. To perform the linear transformation, the input vector is multiplied by a weight matrix and then added to a bias vector. The output of the linear transformation is then passed to a non-linear activation function. The output of the activation function is then passed to the next layer. This process is repeated until the output layer is reached. The output of the output layer is then compared with the target output to calculate the error. The following equation shows the forward propagation process for a single layer:

$$\mathbf{y} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}) \tag{2}$$

where $\mathbf{y}$ is the output of the layer, $\mathbf{x}$ is the input to the layer, $\mathbf{W}$ is the weight matrix, $\mathbf{b}$ is the bias vector, and σ is the activation function.

The activation function is a non-linear function that is applied to the output of the linear transformation. The activation function is responsible for introducing non-linearity to the neural network. The most commonly used activation functions are the sigmoid function, the hyperbolic tangent function, and the rectified linear unit (ReLU). For this project, we use the ReLU activation function. The ReLU activation function is defined by the following equation:

$$\sigma(x) = max(0, x) \tag{3}$$

where x is the input to the activation function.

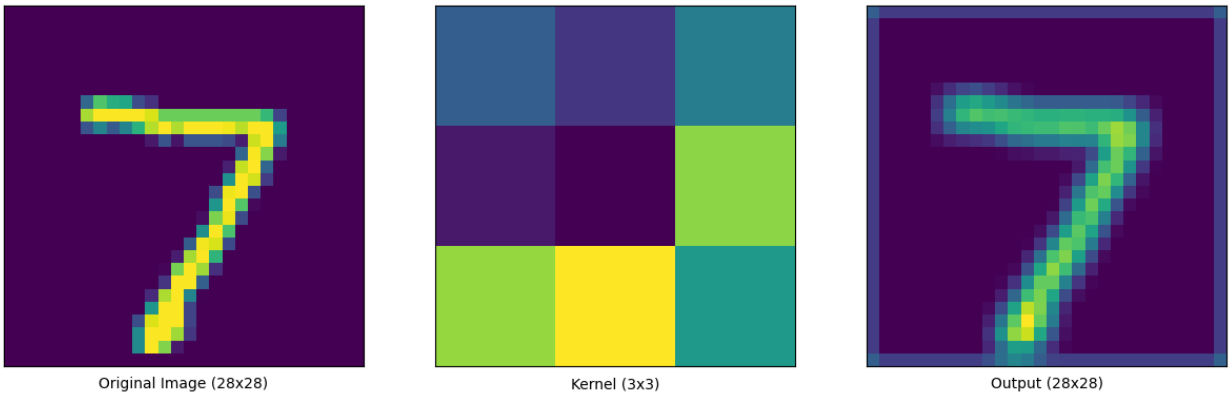

Fig. 2: Sample convolution output of MNIST data with a learned CNN kernel

B. Error Calculation

The error is calculated by comparing the output of the neural network with the target output. The error is then used to adjust the weights of the connections between neurons. For this project, we use the cross-entropy loss function. The cross-entropy loss function is defined by the following equation:

$$L = -\sum_{i=1}^{n} t_i \log(y_i) \tag{4}$$

where L is the loss, t_i is the target output, and y_i is the output of the neural network.

C. Backpropagation

Backpropagation is the process of adjusting the weights of the connections between neurons. The weights are adjusted by calculating the gradient of the loss function with respect to the weights. An optimization algorithm is then used to update the weights. The optimizer used in this project is the Adam optimizer.

The Adam optimizer [3] is an extension of the stochastic gradient descent (SGD) optimizer. The Adam optimizer is defined by the following equations:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{5}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{6}$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{7}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{8}$$

$$w_t = w_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{9}$$

where m_t is the first moment estimate, v_t is the second moment estimate, $\hat{m}_t$ is the bias-corrected first moment estimate, $\hat{v}_t$ is the bias-corrected second moment estimate, w_t is the updated weight, g_t is the gradient of the loss function with respect to the weight, α is the learning rate, β_1 is the exponential decay rate for the first moment estimates, β_2 is the exponential decay rate for the second moment estimates, ϵ is a small constant to prevent division by zero, and t is the iteration number.

The Adam optimizer is an adaptive optimizer. The learning rate is adjusted during the training process. The learning rate is adjusted by calculating the gradient of the loss function with respect to the learning rate. The gradient of the loss function with respect to the learning rate is then used to update the learning rate. The learning rate is updated by the following equation:

$$\alpha_t = \alpha_{t-1} - \frac{\alpha_{t-1}}{2} \frac{dL}{d\alpha} \tag{10}$$

where α_t is the updated learning rate, α_{t-1} is the previous learning rate, and $dL/d\alpha$ is the gradient of the loss function with respect to the learning rate.

IV. Implementation

The model is composed of two convolutional layers and two fully connected layers. The first convolutional layer has 32 filters with a kernel size of 3x3. The second convolutional layer has 64 filters with a kernel size of 3x3. The first fully connected layer has 128 neurons and the second fully connected layer has 10 neurons. The output layer uses a softmax activation function to produce the output.

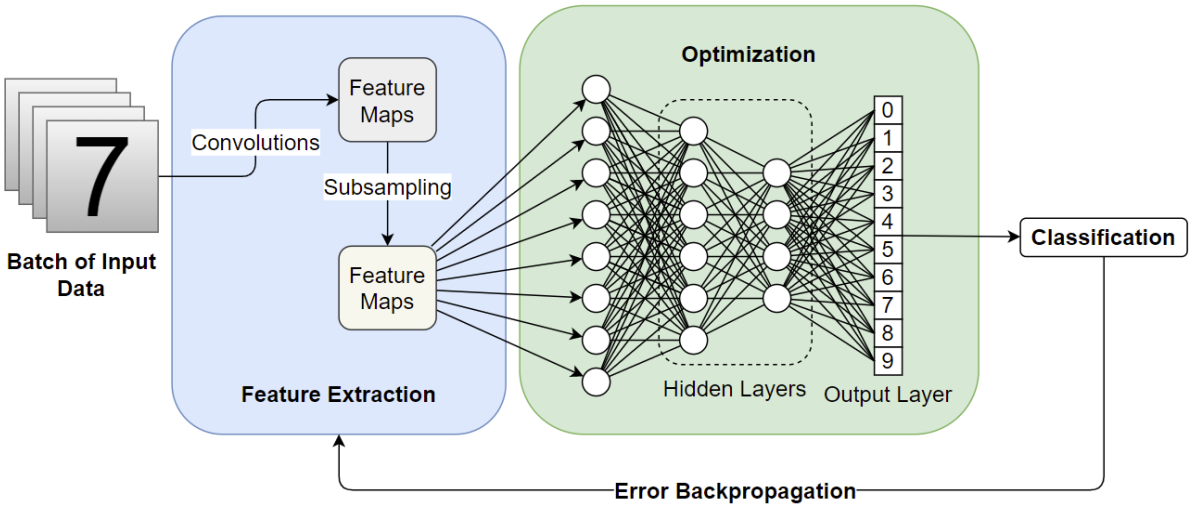

Fig. 4: Convolutional Neural Network for classifying numbers

The model is trained using the Adam optimizer and the cross entropy loss function. The model is trained for 5 epochs with a batch size of 64. The model is evaluated using the accuracy metric.

The following table I shows the loss and accuracy of the model on the test set for both the CNN and the fully connected network.

Model	Loss	Accuracy
CNN	0.03	99.07
Fully Connected	0.10	96.94

TABLE I: Results of the CNN and the fully connected network

The CNN model has a loss of 0.03 and an accuracy of 99.07. The fully connected network has a loss of 0.10 and an accuracy of 96.94. The CNN model has a better accuracy than the fully connected network. This is expected because the CNN model is able to extract features from the input data and use them to classify the input data. The fully connected network is not able to extract features from the input data and is only able to classify the input data based on the raw input data. Additionally, the training and validation curves for the CNN model and FCN model are provided in figures 5 for CNN and 6 for FCN.

The hyperparameters of the CNN model were taken from the PyTorch tutorial on MNIST classification as it is a well-known model and has been tested extensively. The hyperparameters included the batch size, the learning rate, and the optimizer. With the exception of the number of epochs, which was reduced to only 5 in order to reduce the training time.

V. Conclusion

In this project, a CNN model was implemented and trained on the MNIST dataset. The CNN model was able to achieve an accuracy of 99.07 on the test set. The fully connected network was able to achieve an accuracy of 96.94 on the test set. The CNN model was able to achieve a better accuracy than the fully connected network. We also showed the linear algebra behind the convolution operation and how it is used in CNN's. Alongside the training process of a neural network and how the weights of the connections between neurons are adjusted during the training process.

References

[1] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6, 2017.

[2] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[3] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

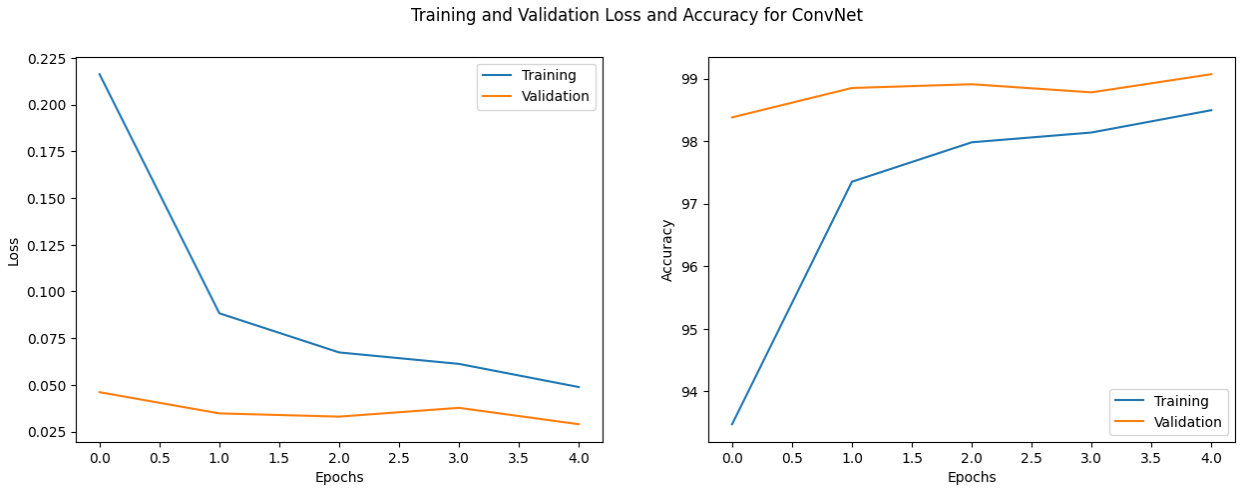

Fig. 5: Convolutional Neural Network training and validation metrics

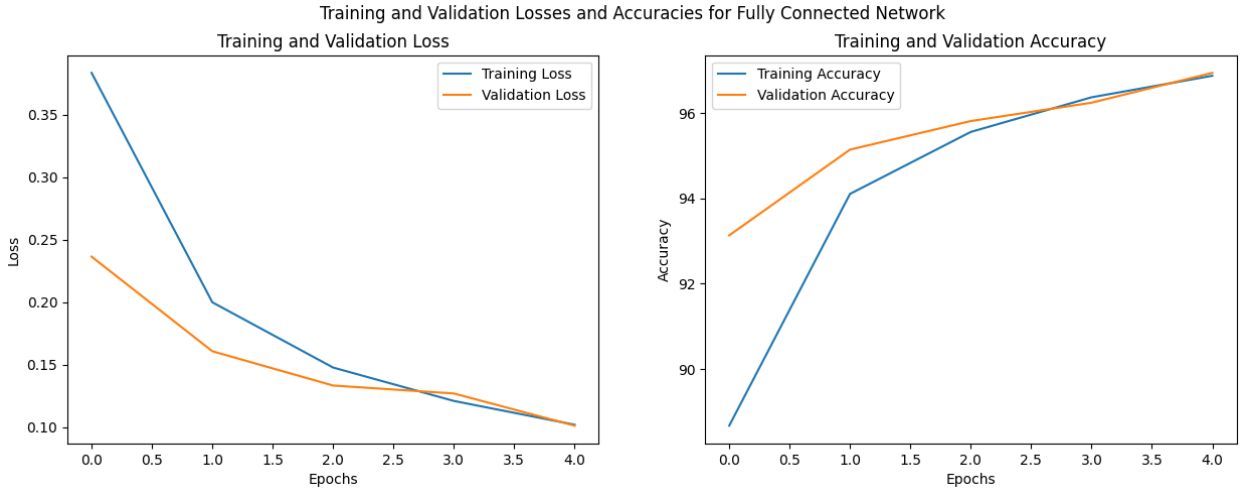

Fig. 6: Fully Connected Neural Network training and validation metrics

[4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[5] R.E. Uhrig. Introduction to artificial neural networks. In *Proceedings of IECON '95 - 21st Annual Conference on IEEE Industrial Electronics*, volume 1, pages 33–37 vol.1, 1995.