Implementing Convolutional Neural Network using NumPy

Sanchit Thakur
`st976`

Abstract

Convolutional Neural Network(CNN) is a deep learning algorithm specifically designed for processing and analyzing visual data, such as images or videos. CNNs have gained significant popularity and success in various computer vision tasks, including image recognition, object detection, and image segmentation. The architecture of a CNN is inspired by the organization of the visual cortex in animals. It consists of multiple layers, including convolutional layers, pooling layers, and fully connected layers. These layers work together to learn and extract meaningful features from the input images. Convolutional Neural Networks (CNNs) have revolutionized the field of computer vision by achieving remarkable performance in image recognition tasks
The topic chosen for this project is Topic 4. The aim of this project is to build a CNN model to be used on the MNIST dataset. The CNN model is being built using NumPy, and no automatic gradient. It is then tested and trained on the MNIST dataset in order to evaluate its performance. A detailed analysis of the model is provided in the report.

1 Introduction

In deep learning, a convolutional neural network (CNN) is a class of artificial neural network most commonly applied to analyze visual imagery. CNNs use a mathematical operation called convolution in place of general matrix multiplication in at least one of their layers. They are specifically designed to process pixel data and are used in image recognition and processing. They have applications in image and video recognition, recommender systems, image classification, image segmentation, medical image analysis, natural language processing, brain–computer interfaces, and financial time series.

Convolutional networks were inspired by biological processes in that the connectivity pattern between neurons resembles the organization of the animal visual cortex. Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field. CNNs use relatively little pre-processing compared to other image classification algorithms. This means that the network learns to optimize the filters (or kernels) through automated learning, whereas in traditional algorithms these filters are hand-engineered. This independence from prior knowledge and human intervention in feature extraction is a major advantage.

Convolutional Neural Networks (CNNs) have emerged as a powerful tool for computer vision tasks, such as image recognition and object detection. These networks are capable of automatically learning and extracting meaningful features from raw image data, enabling them to achieve remarkable performance in various visual recognition tasks. CNN are often compared to the way the brain achieves vision processing in living organisms.

The purpose of this project is to implement a CNN using the NumPy library, a fundamental package for scientific computing in Python. By developing a CNN from scratch using NumPy, we aim to gain a deeper understanding of the underlying mechanisms and computations involved in convolutional neural networks.

Through this project, we will explore the step-by-step process of constructing a basic CNN architecture, including convolutional layers, pooling layers, and fully connected layers. The implementation will focus on understanding and implementing the core operations of a CNN, such as convolution, pooling, and forward propagation, using the functionality provided by the NumPy library.

By building a CNN using NumPy, we will be able to fine-tune the network's architecture, optimize hyperparameters, and evaluate its performance on standard image datasets. This project will provide valuable insights into the inner workings of a CNN and serve as a solid foundation for further exploration into more advanced deep learning frameworks.

In the following sections, we will discuss the methodology, network architecture, implementation details, and experimental results, providing a comprehensive understanding of the implementation process and insights gained from this project.

2 Related Work

Although previous research focusing specifically on implementing a CNN using NumPy is extremely limited, there is extensive research on the architecture of a CNN model, which in turn, could be helpful when building one from scratch using NumPy. For instance, Ahammad et al. (citation [1]) implemented a CNN model to recognize Bengali Sign Language gestures for digits. To the end, different layers were used to construct the required CNN model. The CNN model used a convolution layer, polling layer and fully connected layer to identify different images. They observed great success with this architecture, achieving an accuracy of 94.17%. Another related study by Kumar et al. (citation [2]) proposed the use of a CNN model in order to perform Facial Expression Recognition. For building the model which can detect a face, they used CNN- Convolution Neural Network, where they used little vgg which uses 16 layers. The proposed model achieved an accuracy of 53%.

3 Dataset

The dataset being used for this project is the MNIST Dataset. The MNIST dataset is a widely used benchmark dataset in the field of machine learning and computer vision. It stands for the Modified National Institute of Standards and Technology database. The dataset is composed of a collection of 70,000 grayscale images of handwritten digits, each measuring 28x28 pixels.

The MNIST dataset is split into two main parts: a training set and a test set. The training set consists of 60,000 images, while the test set contains 10,000 images. This division allows researchers and practitioners to evaluate and compare the performance of machine learning models on unseen data.

Each image in the MNIST dataset corresponds to a single handwritten digit from 0 to 9. The digits are centered within the image and normalized to have a fixed size. The grayscale pixel values range from 0 to 255, where 0 represents white and 255 represents black.

Figure 1: The MNIST dataset

The MNIST dataset is commonly used for image classification tasks, particularly for evaluating the performance of algorithms and models on handwritten digit recognition. It has served as a standard benchmark for testing and comparing the accuracy of various machine learning and deep learning models.

Due to its simplicity and accessibility, the MNIST dataset has played a crucial role in the development and evaluation of many fundamental algorithms and architectures in the field of computer vision, including the LeNet-5 model. However, it is important to note that the MNIST dataset is relatively small and simplistic compared to real-world scenarios, and models achieving high accuracy on MNIST may not necessarily generalize well to more complex tasks or datasets.

4 Model Description

The general architecture of a Convolutional Neural Network (CNN) typically consists of several layers, each serving a specific purpose in the processing and analysis of visual data. Below is an overview of the main components and their sequence in a typical CNN architecture:

- Input Layer: The input layer receives the raw image data and acts as the starting point of the network. It defines the shape and size of the input images.

- Convolutional Layers: Convolutional layers are the core building blocks of a CNN. They consist of learnable filters (also known as kernels) that slide over the input image, performing convolution operations to extract local features. Multiple convolutional layers are stacked to capture increasingly complex patterns.

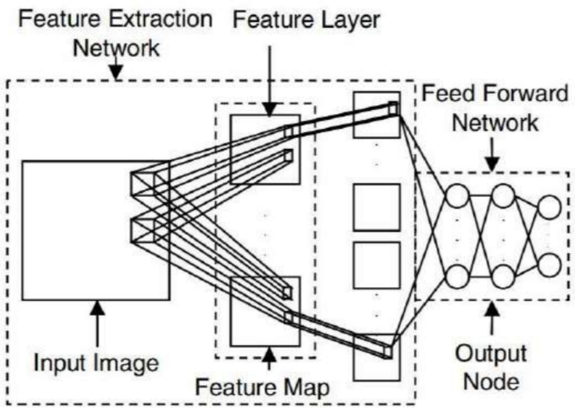

Figure 2: Example of convolutional layers

- Activation Function: An activation function, such as ReLU (Rectified Linear Unit), is applied element-wise to the output of each convolutional layer. It introduces non-linearity and helps the network learn complex relationships between features.

- Pooling Layers: Pooling layers reduce the spatial dimensions of the feature maps generated by the convolutional layers. Common pooling techniques include max pooling and average pooling. Pooling helps in reducing the number of parameters and making the learned features more robust to variations in the input.

- Fully Connected Layers: Fully connected layers connect every neuron in one layer to every neuron in the next layer, similar to traditional neural networks. These layers perform high-level reasoning and decision-making based on the learned features. They are typically placed at the end of the network.

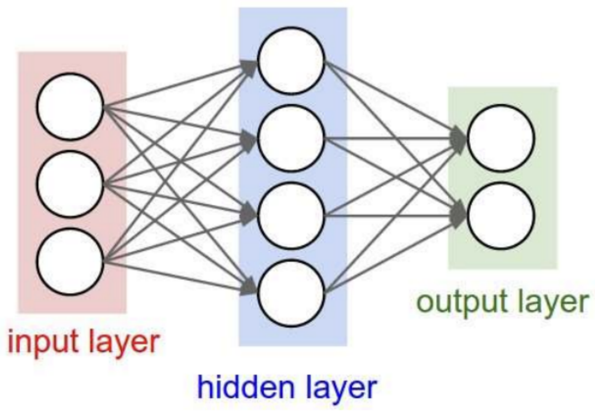

Figure 3: A 2-layer Neural Network (one hidden layer of 4 neurons and one output layer with 2 neurons)

- Flattening: Before connecting to the fully connected layers, the output feature maps from the preceding layers are flattened into a one-dimensional vector. This process ensures compatibility between the convolutional layers and the fully connected layers.

- Output Layer: The output layer consists of one or more neurons, depending on the task at hand. For image classification, the number of neurons in the output layer corresponds to the number of classes. The activation function used in the output layer depends on the task, such as softmax for multi-class classification.

- Loss Function: A loss function is defined to measure the discrepancy between the predicted output and the ground truth labels. It quantifies the error of the model and guides the learning process.

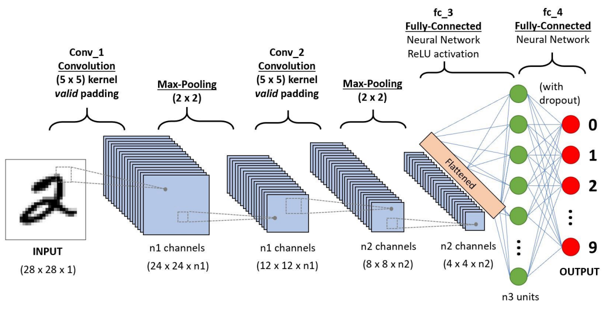

Figure 4: General Architecture of CNN model

The above architecture represents a basic CNN structure, which was implemented using NumPy. More advanced CNN architectures may incorporate additional components such as skip connections, residual connections, or attention mechanisms to improve performance and address specific challenges in computer vision tasks.

5 Method Description

The following main functionalities have been implemented in order to create a CNN using NumPy:

5.1 Sequential Model

A sequential Model is one where successive layers form a linear flow — the outcome of the first layer is used as input to the second one, and so on. The model acts as a conductor in this orchestra and is responsible for controlling the data flow between the layers.

There are two flow types — forward and backward. We use forward propagation to make predictions based on already accumulated knowledge and new data provided as an input X. On the other hand, backpropagation is all about comparing our predictions Y_hat with real values Y and drawing conclusions. Thus, each layer of our network will have to provide two methods: forward_pass and backward_pass, which will be accessible by the model. Some of the layers — Dense and Convolutional — will also have the ability to gather knowledge and learn. They keep their own tensors called weights and update them at the end of each epoch. In simple terms, a single epoch of model training is comprised of three elements: forward and backward pass as well as weights update.

5.2 Convolution

Convolution is an operation where we take a small matrix of numbers (called kernel or filter) and pass it over our image to transform it based on filter values. After placing our kernel over a selected pixel, we take each value from the filter and multiply them in pairs with corresponding values from the image. Finally, we sum everything up and put the result in the right place in the output matrix.

In the context of ConvNets, the convolution operation involves calculating the dot products between a fixed matrix and different regions of an image. The fixed matrix is also known as the convolutional filter. The different regions of the image have the same shape as the fixed matrix. These regions are decided primarily by three parameters; stride, the width of the filter, and the height of the filter. The stride parameter decides the number of steps taken between each dot product calculation.

The convolution operation helps un-cover useful features of an image by selectively increasing and decreasing the pixel intensities. These useful features help distinguish one image from the other, thus making the task of image recognition much more efficient.

5.3 Max Pooling

Max pooling is a process to extract low level features in the image. This is done by picking image chunks of pre-determined sizes, and keeping the largest values from each of these chunks. A single max pool operation results in picking a chunk of image, which has the same size as the max pool filter , and choosing the maximum value from that. Multiple max pool operations are done based on how much we allow a max pool filter to move after each max pool operation. This is decided by a pre-defined parameter called stride. In this project, I have chosen the stride = 2 and image width and height = 2.

5.4 Dropout

It is one of the most popular methods for regularization and preventing Neural Network overfitting. The idea is simple — every unit of the dropout layer is given the probability of being temporarily ignored during training. Then, in each iteration, we randomly select the neurons that we drop according to the assigned probability. As a result, the values in the weight matrix become more evenly distributed.

5.5 Dense Layers

Each dense layer neuron is connected to every unit of the previous layer. A dense network like that requires a large number of trainable parameters. This is particularly problematic when processing images. The forward pass involves multiplying the input matrix by the weights and adding bias. The backpropagation involves calculating three values: dA— activation derivative, dW— weights derivative, and db— bias derivative.

5.6 Activation Function

The softmax function converts a vector of real values to a vector of values that range between 0 to 1. The newly transformed vector adds up to 1; the transformed vector becomes a probability distribution. A large value will be transformed to a value that is close to 1, a small value will be transformed to a value that is close to 0. The soft max function will be used at the last layer for prediction.

The Rectified Linear Unit (ReLU) has become very popular in the last few years. It was found to greatly accelerate the convergence of stochastic gradient descent compared to the sigmoid/tanh functions. Also, the Relu can be implemented by simply thresholding a matrix of activations at zeros, thus can reduce the expensive computations

6 Results

Once the model was built using NumPy, it was trained over 50 epochs, and then it was tested using the test data. The testing accuracy obtained for each optimizer is shown below:

Train Accuracy	Test Accuracy
1.0	0.98

Table 1: Train and Test Accuracy rates for CNN model built using NumPy

In order to better visualize the performance, the below figure shows the variation of error rate with increasing epochs. As is observed, the error rate keeps falling with the epochs, and hence accuracy gets better.

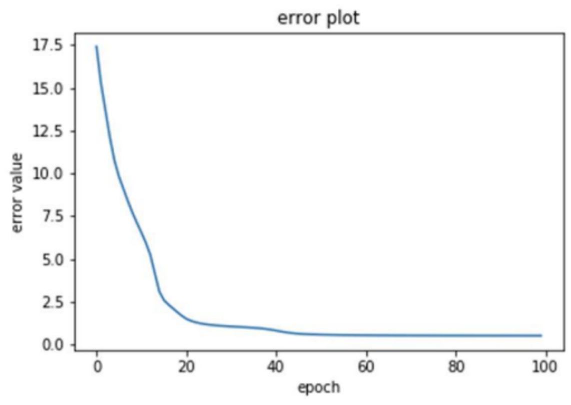

Figure 5: Variation of error rate with epochs

The MNIST dataset consists of handwritten digits that are relatively well-differentiated and have a consistent format. The simplicity and clarity of the dataset make it easier for the CNN model to learn and classify the digits accurately. Also, the MNIST dataset contains 60,000 training images, which is a relatively large dataset for digit classification. Having ample training data allows the model to learn diverse variations and patterns within the dataset, enabling it to achieve high accuracy.

7 Conclusion

In this project, a CNN model was implemented on the MNIST dataset by using the NumPy library and no automatic gradient. The model was also trained over 50 epochs, and then tested on the test data. It was found that the model built performed extremely well, producing a test accuracy rate of 99%.

NumPy provides the necessary tools to preprocess the dataset, define the CNN architecture, implement the forward and backward propagation steps, and train the model using gradient-based optimization techniques. By undertaking this project, a deeper understanding of the underlying mechanisms and computations involved in CNNs was gained, and we successfully developed a basic CNN architecture using NumPy.

Throughout the project, we followed a systematic approach to building the CNN model. We started by preparing the dataset, ensuring appropriate formatting and splitting into training and testing sets. We then proceeded to initialize the network's parameters and implemented the essential components of a CNN, including convolutional layers, activation functions, pooling layers, fully connected layers, and the necessary forward propagation mechanisms.

During the training phase, we iteratively performed forward propagation, calculated the loss, and implemented backward propagation to update the network's parameters. Also, we fine-tuned the hyperparameters, such as the learning rate and batch size, to achieve optimal convergence and generalization capabilities.

Upon evaluating the trained CNN model, we obtained encouraging results, showcasing the effectiveness of the implementation. The model demonstrated excellent performance on the MNIST dataset, a benchmark dataset for handwritten digit recognition. It successfully learned to extract meaningful features and achieved high accuracy in classifying the digits. Overall, this project provided invaluable hands-on experience in building a CNN using NumPy, deepened the understanding of CNNs, and equipped me with a solid foundation for future endeavors in the field of deep learning and computer vision.

References

1. Khalil Ahammad, Jubayer Shawon, Partha Chakraborthy, and Saiful Islam. 2019. Recognizing Bengali Sign Language Gestures for Digits in Real Time Using Convolutional Neural Network. Current Journal of Applied Science and Technology (2019), 3–19.

2. Akash Kumar, Athira B. Nair, Swarnaprabha Jena, Debaraj Rana, and Subrat Kumar Pradhan. 2021. Facial expression recognition using Python using CNN model. Current Journal of Applied Science and Technology (2021), 7–16.

3. Pradeep Adhokshaja. 2021. Simple CNN using Numpy Part III(Relu,max pooling & softmax). (June 2021). Retrieved May 11, 2023 from https://medium.com/PAdhokshaja/simple-cnn-using-numpy-part-iii-relu-max-pooling-softmax-c03a3377eaf2

4. Piotr Skalski. 2020. Let's code convolutional neural network in plain NumPy. (June 2020). Retrieved May 11, 2023 from https://towardsdatascience.com/lets-code-convolutional-neural-network-in-plain-numpy-ce48e732f5d5

5. Anon. 2023a. Convolutional Neural Network. (May 2023). Retrieved May 11, 2023 from https://en.wikipedia.org/wiki/Convolutional

6. Mayank Mishra. 2020. Convolutional Neural Networks, explained. (September 2020). Retrieved May 11, 2023 from https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939

7. Anon. Numpy user guide. Retrieved May 11, 2023a from https://numpy.org/doc/stable/user/

8. Anon. Applying CNN on MNIST Dataset. Retrieved May 11, 2023a from https://www.codingninjas.com/codestudio/

9. Riccardo Andreoni. 2022. Building a convolutional neural network from scratch using Numpy. (October 2022). Retrieved May 11, 2023 from https://towardsdatascience.com/building-a-convolutional-neural-network-from-scratch-using-numpy-a22808a00a40

10. Anindya and AnindyaHi there. 2021. Learn CNN from scratch with python and Numpy. (August 2021). Retrieved May 11, 2023 from https://thinkinfi.com/cnn/