21AIE205 PYTHON FOR MACHINE LEARNING

TERM PROJECT (END SEMESTER)

BATCH A GROUP_4

TEAM MEMBERS	
ABINAYA.N	[CB.EN.U4AIE21001]
DILIP.P	[CB.EN.U4AIE21009]
M.C.S. SANJANA	[CB.EN.U4AIE21029]
SIDDHARTH.D	[CB.EN.U4AIE21064]

Convolutional Neural Networks From Scratch

Table of Contents

Convolutional Neural Networks (CNNs) are a type of deep learning neural network that is particularly well-suited for image and video recognition tasks. They are called "convolutional" because they use a mathematical operation called convolution to automatically and adaptively learn spatial hierarchies of features from input data. CNNs are also used in other tasks, such as natural language processing and audio recognition. These networks are designed to process data with a grid-like topology, such as an image, where the data on one side is connected to data on the other. In this way, CNNs can learn local patterns or features of the images and use them to classify images into different classes.

Convolution on Images

The convolution operation is a crucial component of Convolutional Neural Networks (CNNs) and extracts features from images. The basic idea behind convolution is to take a small matrix called a kernel or filter and slide it over the entire image, element-wise multiplying and summing the image's and kernel's values at each position. The result of this operation is a new matrix called a feature map.

1. Here is a more detailed explanation of how the convolution operation works:
2. The kernel is a small matrix (usually 3x3 or 5x5) used to extract features from the image. It is also called a filter.
3. The kernel is initialized with random values and then learned during training.
4. The kernel is then placed on top of the image, with the top-left corner of the kernel aligned with the top-left corner of the image.
5. The values of the kernel and the image at the current position are element-wise multiplied and summed. This value is then placed in the corresponding position of the feature map.
6. The kernel is then moved one pixel to the right, and the process is repeated. This is called a stride.
7. Once the kernel has been moved over the entire image, the process is repeated for the next layer of the CNN.
8. The convolution operation allows the CNN to extract features from the image, such as edges, corners, and textures, that are important for image classification tasks.

By applying multiple layers of convolutional operation, a CNN can learn increasingly complex features, such as shapes and objects, that are important for image classification.

It is worth noting that the convolution operation is applied to each channel of an image; if it has multiple channels, such as an RGB image, the convolution operation is applied to each channel separately and then combines the results.

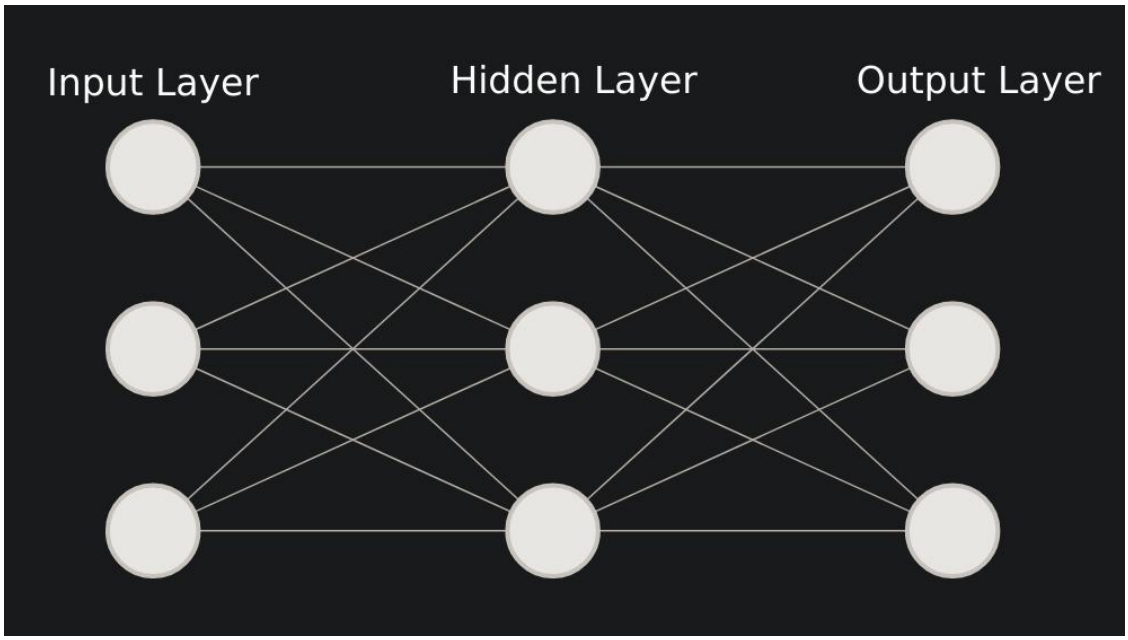
Dense Neural Networks

Before CNNs came into existence, Dense neural networks were used.

Dense Neural Networks, also known as fully connected networks or multi-layer perceptrons (MLPs), were one of the earliest neural networks used for image recognition and other tasks. They are called "dense" because every neuron in one layer is connected to every neuron in the next layer.

The input image is first flattened in a dense neural network, transforming it from a 2D matrix into a 1D array of pixels. This flattened image is then input into the first layer of the network, which is made up of several neurons. Each neuron in this layer takes in the input image, applies a set of weights and biases, and produces an output. This output is then passed on to the next layer, and the process is repeated.

The final layer of the network is the output layer, which produces the final classification of the image. During training, the weights and biases of the neurons are adjusted so that the network can correctly classify the input images.



Dense Neural Network Example

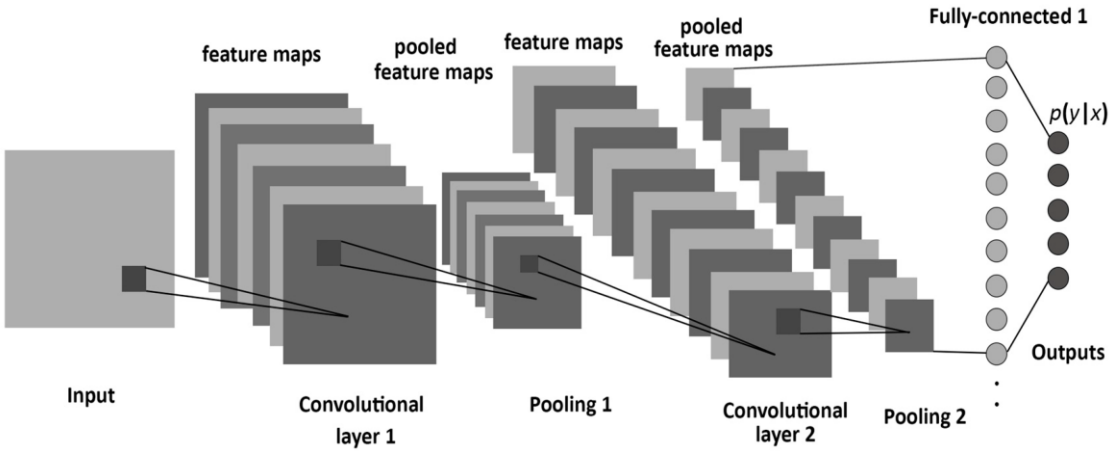
Dense neural networks were widely used in the early days of deep learning. However, they needed to be better suited for image recognition tasks because they needed to consider the spatial structure of the image. This means that they could not effectively learn features such as edges and textures that are important for image classification.

Convolutional Neural Networks (CNNs) were developed to address this issue by using convolutional layers that can learn spatial hierarchies of features and are better suited for image recognition tasks.

Basic Blocks of a CNN model

There are three basic blocks of a CNN model (Figure 2) :

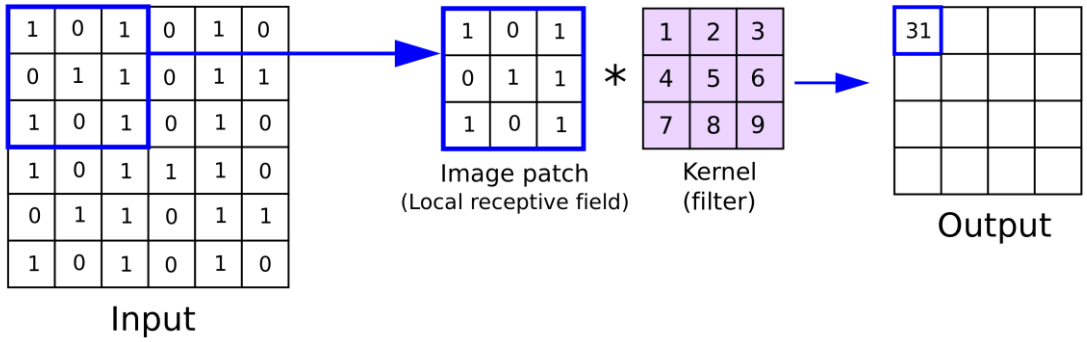
1. Convolutional Layer
2. Pooling Layer
3. Fully Connected Layer (or Dense Layer)



Simple Convolutional Network

Convolutional Layer

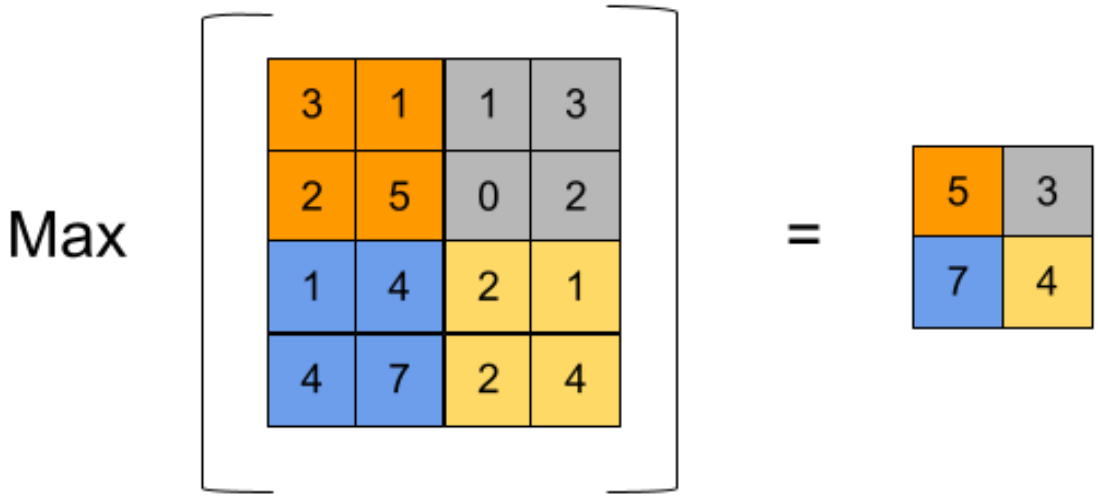
The convolutional layer is the first layer of a CNN. It is made up of several filters, also called kernels. Each filter is a small matrix that is used to extract features from the image. The filters are initialized with random values and then learned during training. The filters are then placed on top of the image, with the top-left corner of the filter aligned with the top-left corner of the image. The values of the filter and the image at the current position are element-wise multiplied and summed. This value is then placed in the corresponding position of the feature map. The filter is then moved one pixel to the right, and the process is repeated. Once the filter has been moved over the entire image, the process is repeated for the next filter. The convolution operation allows the CNN to extract features from the image, such as edges, corners, and textures, that are important for image classification tasks (Figure 3).



Sample Convolution Operation On a Patch

Pooling Layer

The pooling layer is used to reduce the spatial size of the feature map produced by the convolutional layer. This is done by dividing the feature map into non-overlapping regions and computing a summary statistic for each region, such as the max or average. This operation results in a new feature map with a smaller spatial size. The pooling layer is used to reduce the number of parameters in the network, which reduces the amount of computation required and helps prevent overfitting (Figure 4).



Max Pooling

Fully Connected Layer (or Dense Layer)

The fully connected layer is the last layer of a CNN. It comprises a number of neurons, each of which takes in the previous layer's output and produces an output. The output of the fully connected layer is then passed on to the output layer, which produces the final classification of the image. During training, the weights and biases of the neurons are adjusted so that the network can correctly classify the input images.

Implementation

Libraries Info

1. Numpy - Used for all the linear algebra operations.
2. KerasDatasets - Used to load the dataset.

Classes

Convolutional Layer

This is the class where the convolutional layer is implemented.

Input Parameters - Number of Kernels, Kernel Size

The methods in this class are discussed below:

patches_generator

- The convolution operation is performed on the image patches, which are generated using this method. Generally, convolution is performed on the entire image, but in this case, the image is divided into patches, and then convolution is performed on each patch. Here, each patch will correspond to one value in the convolved output.

forward_prop

- This method performs the forward propagation of the convolution operation. The input image is divided into patches, and then the convolution operation is performed on each patch.

back_prop

- This method performs the backpropagation of the convolution operation. The gradients of the loss with respect to the weights and biases of the convolutional layer are computed. The gradients of the loss with respect to the input image are also computed. This is done by computing the gradients of the loss with respect to the output of the convolutional layer and then using the chain rule to compute the gradients of the loss with respect to the input image.

Pooling Layer

This is the class where the pooling layer is implemented.

Input Parameters - Pooling Size

The methods in this class are discussed below:

patches_generator

- The pooling operation is performed on the image patches generated using this method. Generally, pooling is performed on the entire image, but in this case, the image is divided into patches, and then pooling is performed on each patch. Here, each patch will correspond to one value in the pooled output.

forward_prop

- This method performs the forward propagation of the pooling operation. The input image is divided into patches, and then the pooling operation is performed on each patch. The pooling operation is performed by computing the max value in each patch.

back_prop

- Here, there are no weights and biases to be updated, but the output is needed to update the weights of the convolutional layer.

Softmax Layer

The softmax function is a generalization of the logistic sigmoid function for the case of multiple classes. It takes an input vector and returns a probability distribution over the classes. The input vector can be any real-valued vector, and the output is a probability vector that sums to 1.

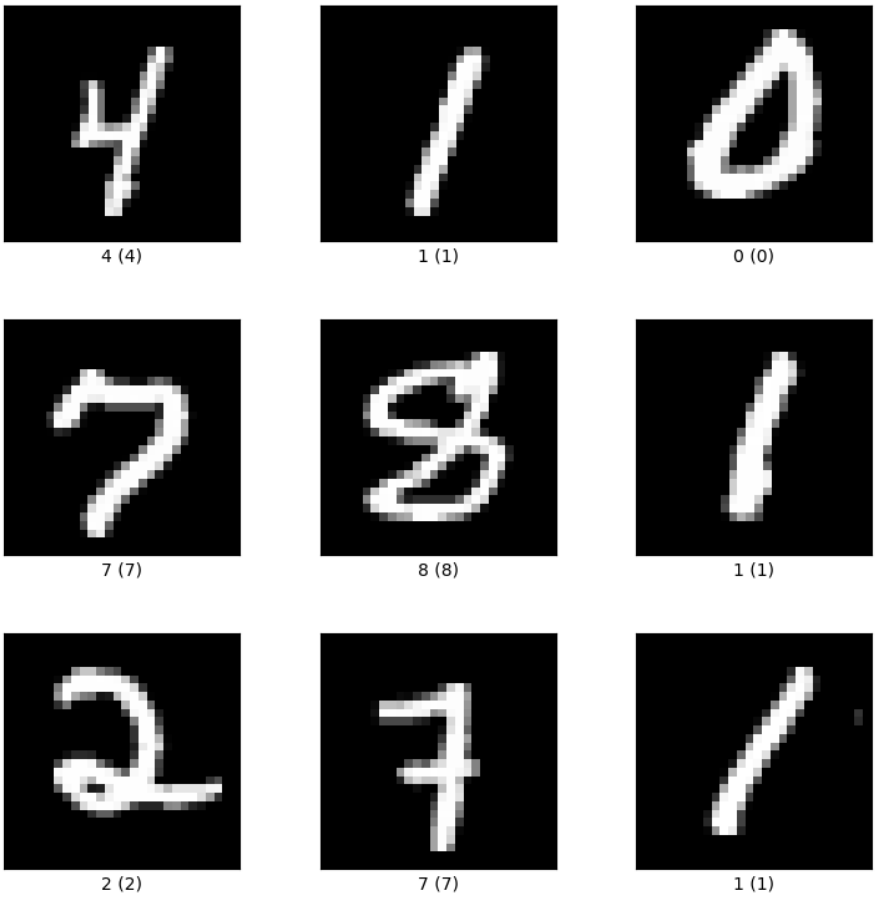
The essential idea softmax function is to normalize the exponential of the input vector so that the sum of all the elements in the output vector is 1. This allows the function to be used as a prediction function that assigns a probability to each class.

Data Pipeline

To check our CNN, we use two datasets:

MNIST Dataset

This is a dataset of handwritten digits. It contains 60,000 training images and 10,000 test images. Each image is 28x28 pixels and is grayscale. The dataset is available in the Keras datasets module. We directly use this dataset to train our CNN. This dataset does not need any preprocessing, as it is already in the required format.



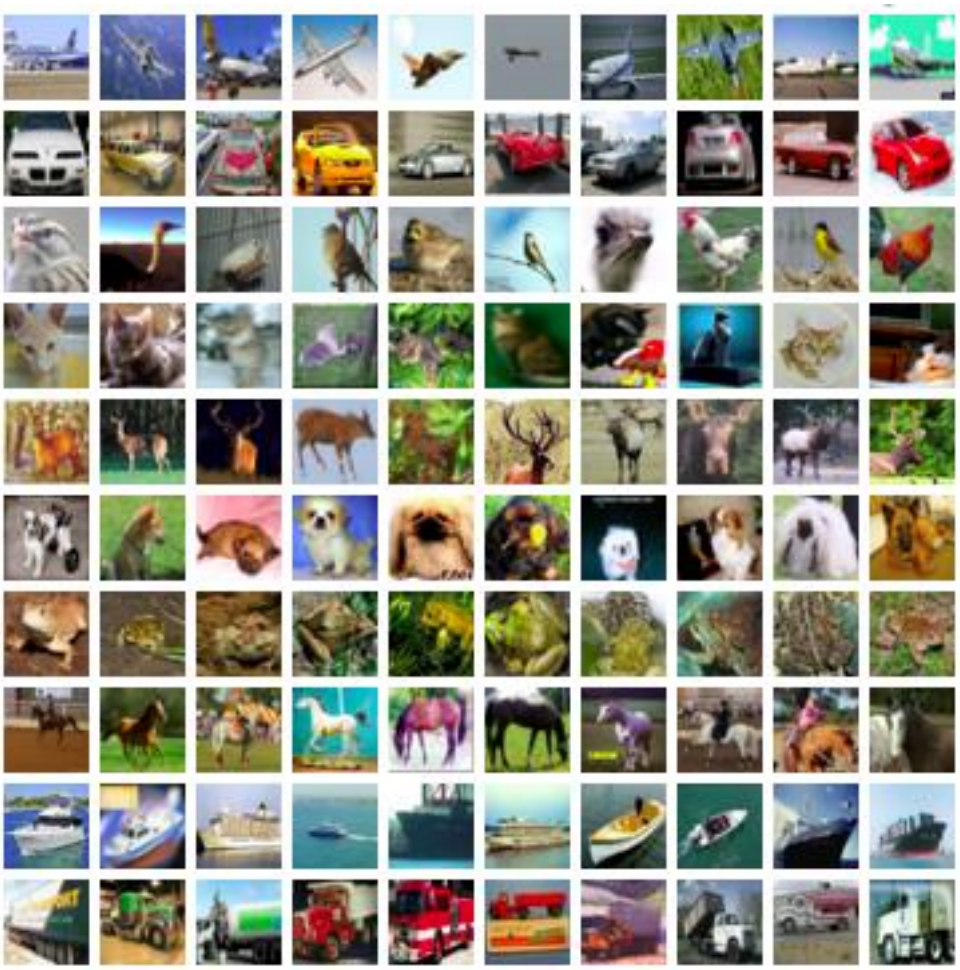
MNIST Samples

Preprocessing MNIST Dataset

The input images are scaled down to the 0-1 range. Since we used Keras to import our dataset, the labels were already one-hot encoded. Images were already in greyscale, so no further preprocessing was required.

CIFAR 10 Dataset

This is a dataset of 10 different classes of images. It contains 50,000 training images and 10,000 test images. Each image is 32x32 pixels and is RGB. The dataset is available in the Keras datasets module. The preprocessing we do here is to convert the images to grayscale, as the CNN we have implemented works with one channel.



CIFAR-10 Samples

Preprocessing CIFAR 10 Dataset

The input images are scaled down to the 0-1 range. Since we used keras to import our dataset, the labels were already one-hot encoded. The images are converted to grayscale by taking each pixel's mean of the RGB values. This is done because the CNN we have implemented works with one channel.

Training

We created a simple CNN model consisting of 1 convolutional layer, one pooling layer, and the prediction head. The model was trained on the MNIST and CIFAR 10 datasets for one epoch.

It took around 1.5 minutes to train 1000 images for the MNIST dataset and around 15 min to train 1000 images for the CIFAR 10 dataset.

It took 2 hours to train the MNIST model and 20 min to test the MNIST model. In the case of CIFAR, it took 2 hours to train the model and 20 min to test the model.

Results

On the MNIST dataset, we got an accuracy of ~90%. However, on CIFAR 10, we got only around 30% accuracy. The results can be improved by creating deeper CNN models and training them for more epochs. This could not be done because of the limitations discussed below.

Dataset	Training Accuracy	Test Accuracy	Prescision	Recall	F1 Score
MNIST	0.90	0.90	0.90	0.90	0.90
CIFAR 10	0.30	0.30	0.30	0.30	0.30

Limitations

1. Since the entire code was made with NumPy, there is no scope for parallelization. This makes the training extremely slow. Modern deep learning frameworks use GPUs for parallelization, which makes the training much faster. Additionally, some frameworks take advantage of hardware-level optimizations for commonly used operations like convolutions and pooling. This makes the training even faster.
2. The current implementation trains image by image. This is because having the batched dataset would mean that the convolution kernel would have to make nested loops to compute the convolution operation. This would make the code extremely slow and, in the worse case, fill up the existing memory and crash the code.
3. The current implementation works only with one channel. The reason is the same as for batched inputs.

Using Deep Learning Framework - Tensorflow

To show the power of modern deep learning frameworks, we have implemented a slightly more complex CNN model and trained it using GPU.

Training setup:

- Learning Rate - 0.001
- Optimizer - Adam
- Loss Function - Sparse Categorical Crossentropy
- Batch Size - 64

Here, we were able to achieve the following results.

Dataset	Train Accuracy	Test Accuracy	Presicion	Recall	F1 Score
MNIST	0.9894	0.9852	0.9854	0.9852	0.9852
CIFAR 10	0.6785	0.6336	0.6336	0.6439	0.6308

Conclusion

In this project, we explored the inner workings of Convolutional Neural Networks. We implemented a simple, barebones CNN, which can be used to classify images. We also explored the limitations of the current implementation and discussed how these limitations could be overcome to make CNN more efficient. We also discussed the importance of using modern deep learning frameworks, which use hardware-level optimizations, to make the training faster.

We have also included an implementation using deep learning frameworks to show how efficient they are compared to NumPy.