Application of CNN on MNIST Dataset

Summer Project 2021

Indian Institute of Technology Kanpur

July31,2021

By:

Arkonil Dhar Saumyadip Bhowmick Shreya Pramanik

Shubha Sankar Banerjee Souvik Bhattacharyya

Data Description: The MNIST handwritten digit classification problem is a standard dataset used in computer vision and deep learning. The MNIST dataset is an acronym that stands for the Modified National Institute of Standards and Technology dataset. Our unrotated dataset has a training set of 50,000 and a test set of 10000 small square 28×28 pixel grayscale images of handwritten single digits varying from 0 to 9. And our rotated dataset has 12000 train data and 10000 test data.

We developed convolutional neural networks from scratch to classify a given image of a handwritten digit into one of 10 classes representing integer values from 0 to 9. Then used the same CNN model for testing the accuracy of the merged dataset.

Convolutional Neural Network:

Introduction:

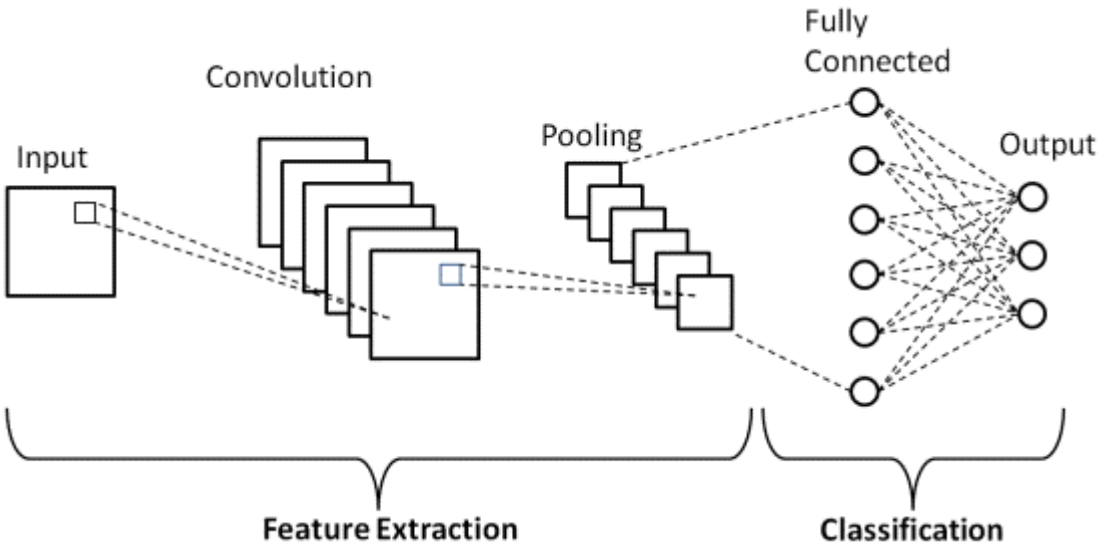
CNNs are a class of Deep Neural Networks that can classify features from images and are widely used for image classification, medical image analysis and computer vision. The term 'Convolution" in CNN denotes the mathematical function of convolution which is a special kind of linear operation wherein two functions are multiplied to produce a third function which expresses how the shape of one function is modified by the other i.e. two images which can be represented as matrices are multiplied to give an output that is used to extract features from the image.

Basic CNN Architecture:

This architecture is composed of two main blocks:

● The first block is made of several convolution kernels which filter the image and return “feature maps”, which are then normalized (with an activation function) or resized. We filter the features maps obtained again with new kernels, which give us new features maps to normalize and resize, then we can filter again, and so on. Finally, the values of the last feature maps are concatenated into a vector. This vector defines the output of the first block and the input of the second.

● The second block is a fully connected layer used for classification. The input vector values are transformed with several linear combinations and activation functions to return a new vector to the output. The no of elements in the last vector are same as no of classes. Element i represents the probability that the image belongs to class i. Each element is therefore between 0 and 1, and the sum of all is worth 1. These probabilities are calculated by the last layer of this block using softmax activation function for multiclass classification.

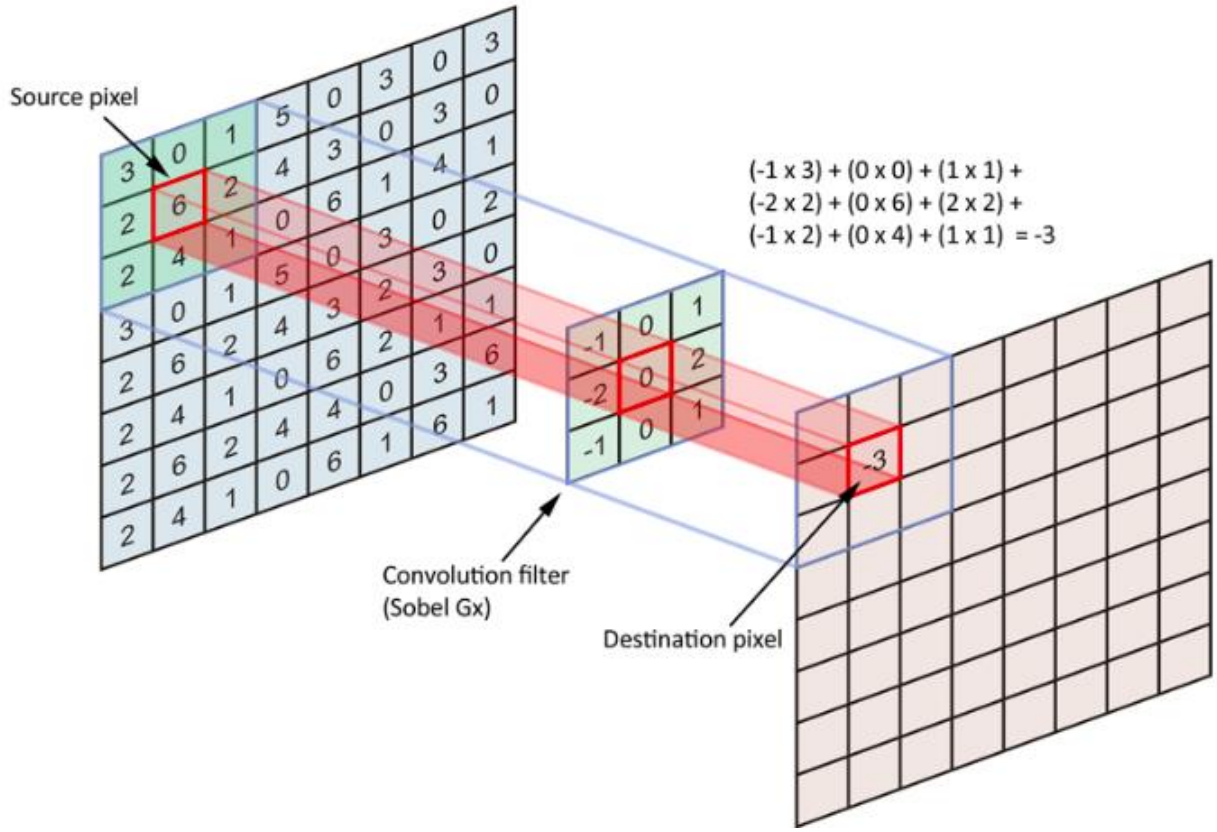


Different Layers of CNN:

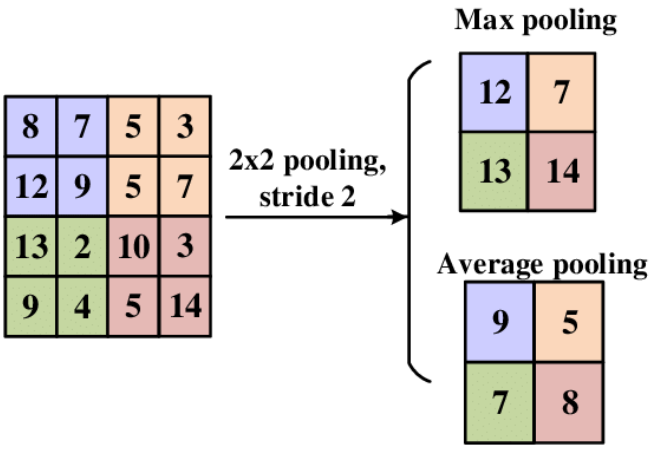
CNNs are made up stacking three main layers: convolutional layers, pooling layers, fully connected (FC) layers. In addition to these there is also dropout layer.

●**Convolutional Layer:** This is the first layer that is used to extract the various features from the input images. This is done by convolution filtering: the principle is to perform convolution between the input image and the filter of a particular size MxM. By sliding the filter over the input image, the dot product is taken between the filter and the parts of the input image with respect to the size of the filter (MxM). The convolutional layer thus receives several images as input, and calculates the convolution of each of them with each filter.

The output is termed as the Feature map which gives us information about the image such as the corners and edges: the higher the value, the more the corresponding place in the image resembles the feature. Later, this feature map is fed to other layers to learn several other features of the input image.



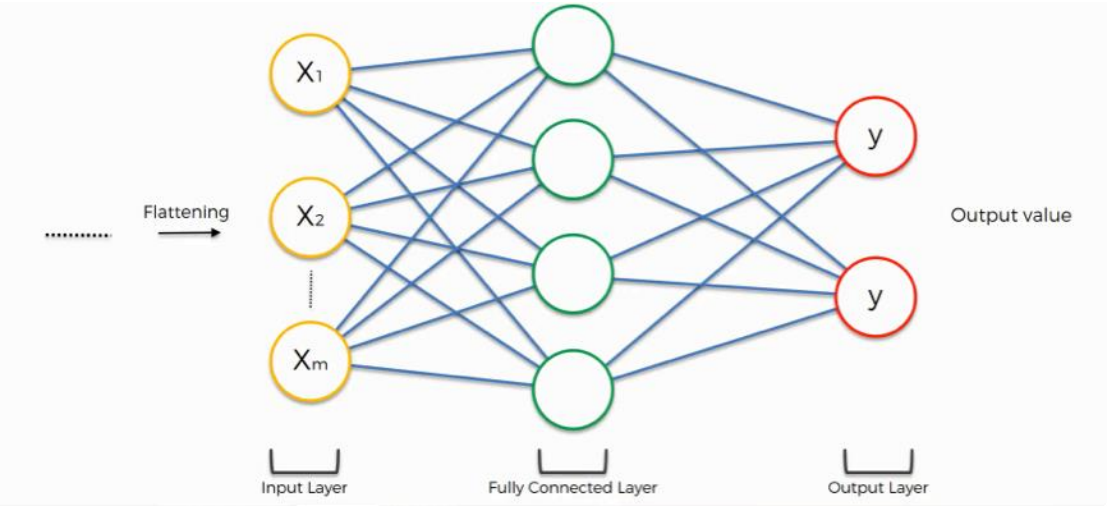
●**Pooling Layer**: In most CNNs, a Convolutional Layer is followed by a Pooling Layer. The aim is to decrease the size of the convolved feature map thus reducing no of parameters and calculations in the network. Depending upon method used, there are several types of Pooling operations. In Max Pooling, the largest element is taken from feature map. Average Pooling calculates the average of the elements in a predefined sized Image section. The total sum of the elements in the predefined section is computed in Sum Pooling. The Pooling Layer usually serves as a bridge between the Convolutional Layer and the FC Layer.



●**Fully Connected Layer:** The fully-connected layer is always the last layer of a neural network, convolutional or not — so it is not characteristic of a CNN. The FC layer consists of the weights and biases along with the neurons and is used to connect the neurons between two different layers. The input image from the previous layers is flattened and fed to the FC layer. The flattened vector then undergoes few more FC layers where the mathematical functions operations usually take place.

The last fully-connected layer classifies the image as an input to the network: it returns a vector of size N, where N is the number of classes in our image classification problem. Each element of the vector indicates the probability for the input image to belong to a class. To calculate the probabilities, the fully-connected layer, therefore, multiplies each input element by weight, makes the sum, and then applies an activation function (sigmoid if N=2, Softmax if N>2). The fact that each input value is connected with all output values explains the term fully-connected.

In neural network to minimize the error we use loss function.



®®®**Activation Function:** one of the most important parameters of the CNN model is the activation function. They are used to learn and approximate any kind of continuous and complex relationship between variables of the network. In simple words, it decides which information of the model should fire in the forward direction and which ones should not at the end of the network.

It adds non-linearity to the network. There are several commonly used activation functions such as the ReLU, Softmax, tanH and the Sigmoid functions. Each of these functions have a specific usage. For a binary classification CNN model, sigmoid and softmax functions are preferred and for a multi-class classification, generally softmax us used.

®®®**Loss Function**: The Loss Function is one of the important components of Neural Networks. Loss is nothing but a prediction error of Neural Network and the method to calculate the loss is called Loss Function.In simple words, the Loss is used to calculate the

gradients and the gradients are used to update the weights of the Neural Network. There are several loss functions:

◦Mean Squared Error (MSE)

◦Binary Crossentropy (BCE)

◦Categorical Crossentropy (CC)

◦Sparse Categorical Crossentropy (SCC)

Last-layer activation and loss function combinations:

Problem type	Last-layer activation	Loss function	Example
Binary classification	sigmoid	binary_crossentropy	Dog vs cat, Sentiemnt analysis(pos/neg)
Multi-class, single-label classification	softmax	categorical_crossentropy	MNIST has 10 classes single label (one prediction is one digit)
Multi-class, multi-label classification	sigmoid	binary_crossentropy	News tags classification, one blog can have multiple tags
Regression to arbitrary values	None	mse	Predict house price(an integer/float point)
Regression to values between 0 and 1	sigmoid	mse or binary_crossentropy	Engine health assessment where 0 is broken, 1 is new

●**Dropout Layer:** Usually, when all the features are connected to the FC layer, it can cause overfitting in the training dataset. Overfitting occurs when a particular model works so well on the training data but fails to perform satisfactorily when used on a new data.

To overcome this problem, a dropout layer is utilised wherein a few neurons are dropped from the neural network during training process resulting in reduced size of the model. On passing a dropout of 0.3, 30% of the nodes are dropped out randomly from the neural network.

Hyperparameter Tuning in CNN: Often our model tends to overfit resulting in poor accuracy of test dataset which pushes us to find a way for building best possible model for our data. To solve this problem, we resort to hyperparameter tuning.

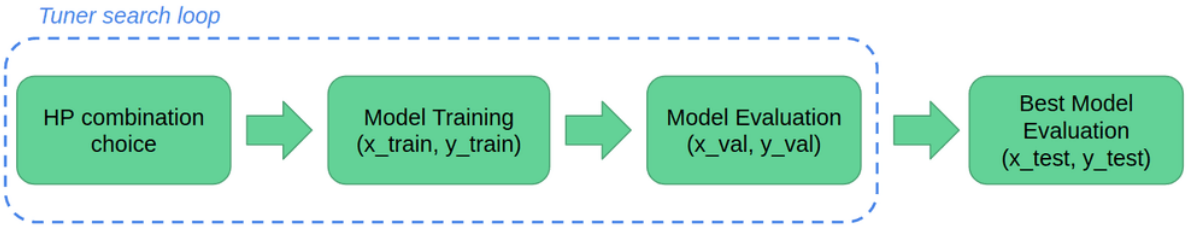
A machine learning model has two types of parameters:

◦***Trainable parameters*****,** which are learned by the algorithm during training. For instance, the weights of a neural network are trainable parameters.

◦***Hyperparameters***, which need to be set before launching the learning process. The learning rate or the number of units in a dense layer are hyperparameters.

Hyperparameters can be numerous even for small models. Tuning them can be a real brain teaser but worth the challenge: a good hyperparameter combination can highly improve model's performance. An easy way to perform this is using keras-tuner.

So how does keras-tuner work?



First, a tuner is defined. Its role is to determine which hyperparameter combinations should be tested. The library search function performs the iteration loop, which evaluates a certain number of hyperparameter combinations. Evaluation is performed by computing the trained model's accuracy on a held-out validation set.

Finally, the best hyperparameter combination in terms of validation accuracy can be tested on a held-out test set.

The most intuitive way to perform hyperparameter tuning is to randomly sample hyperparameter combinations and test them out. This is exactly what the ***RandomSearch tuner*** does. We have used RandomSearch as our hyperparameter tuning method for MNIST classification here.