Image Processing using CNN: A beginners guide

M Mohit
Last Updated : 31 May, 2024

25

Introduction

The various deep learning methods use data to train neural network algorithms to do a variety of machine learning tasks, such as the classification of different classes of objects. Convolutional neural networks are deep learning algorithms that are very powerful for the analysis of images. This article will explain to you how to construct, train and evaluate convolutional neural networks and MNIST dataset.

You will also learn how to improve their ability to learn from data, and how to interpret the results of the training. Deep Learning has various applications like image processing, natural language processing, etc. It is also used in Medical Science, Media & Entertainment, Autonomous Cars, etc. In this, article you will get understanding for Cnn for beginners, how CNN algorithm for image processing is you can do . Also, this article we clarify all about the CNN and image processing.So lets begin with Cnn and image processing for cnn.

This article was published as a part of the Data Science Blogathon

What is an Image?

Images contain data of RGB combination. Matplotlib can be used to import an image into memory from a file. The computer doesn't see an image, all it sees is an array of numbers. Color images are stored in 3-dimensional arrays. The first two dimensions correspond to the height and width of the image (the number of pixels). The last dimension corresponds to the red, green, and blue colors present in each pixel.

So from this, we can infer that an image in the context of this article refers to a 2D array or matrix of pixel values representing visual data, where each pixel has a specific color value composed of red, green, and blue intensities.

What is Image Processing?

The various deep learning methods use data to train neural network algorithms to do a variety of machine learning tasks, such as the classification of different classes of objects. Convolutional neural networks are deep learning algorithms that are very powerful for the analysis of images. This article will explain to you how to construct, train and evaluate convolutional neural networks.

So image processing in this context refers to the analysis, manipulation, or extraction of information from image data using techniques like convolutional neural networks.

Types of Image Processing

The article focuses specifically on using convolutional neural networks (CNNs) for image recognition and classification tasks. Some key types of image processing covered are:

- Object recognition/classification
- Handwritten digit recognition (using MNIST dataset)
- Feature extraction from images using convolutional and pooling layers

While it doesn't explicitly list out other types, CNNs can also be used for tasks like:

- Object detection and localization
- Image segmentation
- Image retrieval
- Image denoising/restoration
- Image super-resolution

So the main type covered in-depth is using CNNs for image recognition and classification problems, which falls under the broader umbrella of image processing and computer vision applications.

What is CNN?

CNN is a powerful algorithm for image processing. These algorithms are currently the best algorithms we have for the automated processing of images. Many companies use these algorithms to do things like identifying the objects in an image.

Images contain data of RGB combination. Matplotlib can be used to import an image into memory from a file. The computer doesn’t see an image, all it sees is an array of numbers. Color images are stored in 3-dimensional arrays. The first two dimensions correspond to the height and width of the image (the number of pixels). The last dimension corresponds to the red, green, and blue colors present in each pixel.

Also Read- [How Do Computers Store Images?]

Three Layers of CNN

Convolutional Neural Networks specialized for applications in image & video recognition. CNN is mainly used in image analysis tasks like Image recognition, Object detection & Segmentation.

There are three types of layers in Convolutional Neural Networks:

Convolutional Layer

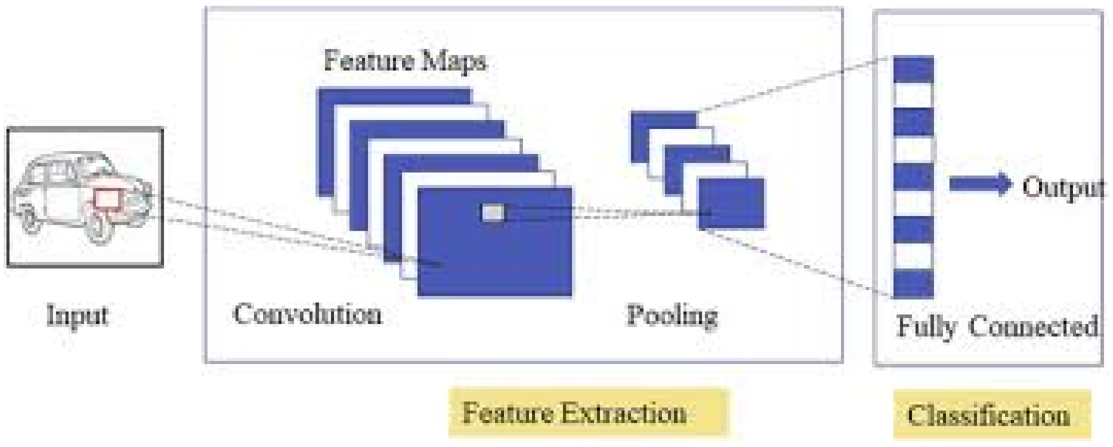
In a typical neural network each input neuron is connected to the next hidden layer. In CNN, only a small region of the input layer neurons connect to the neuron hidden layer.

Pooling Layer

The pooling layer is used to reduce the dimensionality of the feature map. There will be multiple activation & pooling layers inside the hidden layer of the CNN.

Fully-Connected layer

Fully Connected Layers form the last few layers in the network. The input to the fully connected layer is the output from the final Pooling or Convolutional Layer, which is flattened and then fed into the fully connected layer.



Also Read- [Introduction to Convolutional Neural Networks (CNN)]

MNIST Dataset

In this article, we will be working on object recognition in image data using the MNIST dataset for handwritten digit recognition.

The MNIST dataset is a widely-used benchmark dataset in the field of machine learning and computer vision. It consists of a large collection of handwritten digit images, commonly used for training and evaluating image processing systems like convolutional neural networks (CNNs).

Specifically, the MNIST dataset contains 70,000 small square 28×28 pixel grayscale images of handwritten digits from 0 to 9. These images were collected from among Census Bureau employees and high school students who had familiarity with reading census forms.

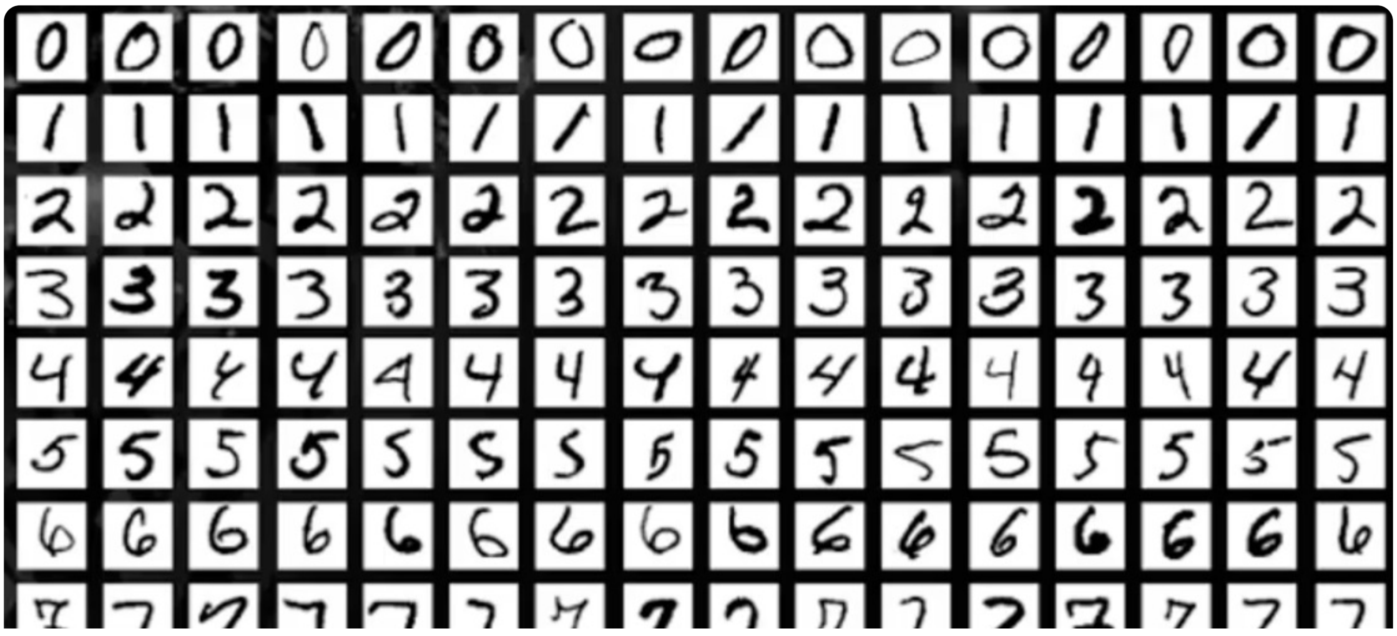
The full dataset is divided into two parts

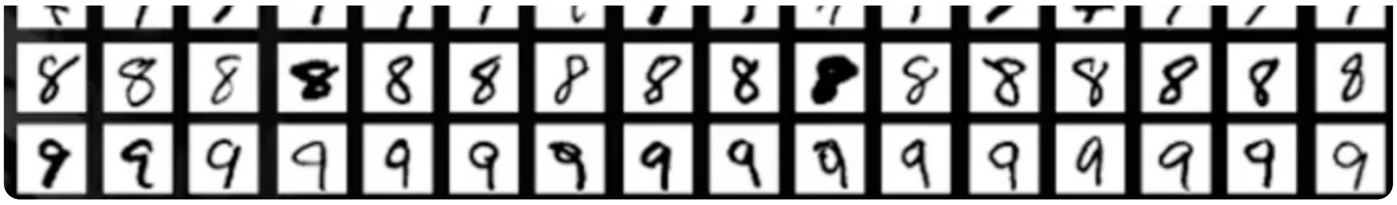
- Training Set: 60,000 images, used to train machine learning models to recognize handwritten digit patterns.
- Test Set: 10,000 images, used to evaluate the performance of models trained on the training set.

Each image in the MNIST dataset is associated with a label, indicating the digit (0-9) that the image represents. The goal is to train a machine learning model, like a CNN, to accurately classify or predict the digit shown in each image based on the pixel patterns.

Despite its simplicity, the MNIST dataset serves as an excellent starting point and benchmark for developing and testing image classification algorithms. Its manageable size and relatively simple problem space make it ideal for educational purposes and prototyping new techniques before advancing to more complex image datasets.

By working through the MNIST dataset examples in this article, you will gain hands-on experience with implementing convolutional neural networks for image recognition tasks, a foundational step towards more advanced computer vision applications.





Loading the MNIST Dataset

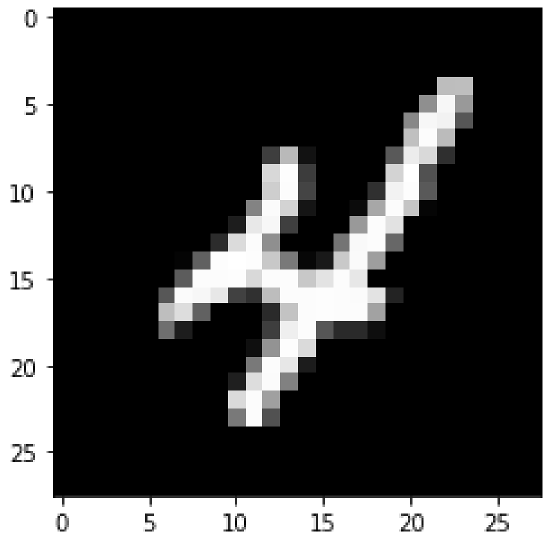
Install the TensorFlow library and import the dataset as a train and test dataset.

```
import matplotlib.pyplot as plt
from keras.datasets import mnist

# Load the MNIST dataset
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# Plot the sample output of the image
plt.imshow(X_train[9], cmap=plt.get_cmap('gray'))
plt.show()
```

Output:



Deep Learning Model with Multi-Layer Perceptrons using MNIST

In this model, we will build a simple neural network model with a single hidden layer for the MNIST dataset for handwritten digit recognition.

A perceptron is a single neuron model that is the basic building block to larger neural networks. The multi-layer perceptron consists of three layers i.e. input layer, hidden layer and output layer. The hidden layer is not visible to the outside world. Only the input layer and output layer is visible. For all DL models data must be numeric in nature.

Also Read-[Deep Learning Project to Recognize Handwritten Digit]()

Step-1: Import key libraries

Copy Code

```
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import np_utils
```

Step-2: Reshape the data

Each image is 28X28 size, so there are 784 pixels. So, the output layer has 10 outputs, the hidden layer has 784 neurons and the input layer has 784 inputs. The dataset is then converted into float.

Copy Code

```
# Determine the number of pixels
number_pix = X_train.shape[1] * X_train.shape[2]

# Reshape the data and convert to float32
X_train = X_train.reshape(X_train.shape[0], number_pix).astype('float32')
X_test = X_test.reshape(X_test.shape[0], number_pix).astype('float32')
```

Step-3: Normalize the data

NN models usually require scaled data. In this code snippet, the data is normalized from (0-255) to (0-1) and the target variable is one-hot encoded for further analysis. The target variable has a total of 10 classes (0-9)

```
from keras.utils import to_categorical

X_train /= 255
X_test /= 255

y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

num_classes = y_train.shape[1]
print(num_classes)
```

Output:10

Now, we will create an NN_model function and compile the same

Step-4: Define the model function

```
def nn_model():model = Sequential()model.add(Dense(number_pix, input_dim=nu
```

There are two layers one is a hidden layer with activation function ReLu and the other one is the output layer using the softmax function.

Step-5: Run the model

```
model = nn_model()model.fit(X_train, y_train, validation_data=(X_test, y_te
```

Output:Epoch 1/10 300/300 – 11s – loss: 0.2778 – accuracy: 0.9216 – val_loss: 0.1397 – val_accuracy: 0.9604 Epoch 2/10 300/300 – 2s – loss: 0.1121 – accuracy: 0.9675 – val_loss: 0.0977 – val_accuracy: 0.9692 Epoch 3/10 300/300 – 2s – loss: 0.0726 – accuracy: 0.9790 – val_loss: 0.0750 – val_accuracy: 0.9778 Epoch 4/10 300/300 – 2s – loss: 0.0513 – accuracy: 0.9851 – val_loss: 0.0656 – val_accuracy: 0.9796 Epoch 5/10 300/300 – 2s – loss: 0.0376 – accuracy: 0.9892 – val_loss: 0.0717 – val_accuracy: 0.9773 Epoch 6/10 300/300 – 2s – loss: 0.0269 – accuracy: 0.9928 – val_loss: 0.0637 – val_accuracy: 0.9797 Epoch 7/10 300/300 – 2s – loss:

0.0208 – accuracy: 0.9948 – val_loss: 0.0600 – val_accuracy: 0.9824 Epoch 8/10 300/300 – 2s – loss: 0.0153 – accuracy: 0.9962 – val_loss: 0.0581 – val_accuracy: 0.9815 Epoch 9/10 300/300 – 2s – loss: 0.0111 – accuracy: 0.9976 – val_loss: 0.0631 – val_accuracy: 0.9807 Epoch 10/10 300/300 – 2s – loss: 0.0082 – accuracy: 0.9985 – val_loss: 0.0609 – val_accuracy: 0.9828 The error is: 1.72%

In the model results, it is visible as the number of epochs increases the accuracy improves. The error is 1.72%, lower the error higher the accuracy of the model.

Convolutional Neural Network Model using MNIST

In this section, we will create simple CNN models for MNIST that demonstrate Convolutional layers, Pooling layers & Dropout layers.

Step-1: Import all the necessary libraries

```
import numpy as npfrom keras.models import Sequentialfrom keras.layers impo
```

Step-2: Set the seed for reproducibility and load the data MNIST data

```
seed=10 np.random.seed(seed) (X_train,y_train), (X_test, y_test)= mnist.loa
```

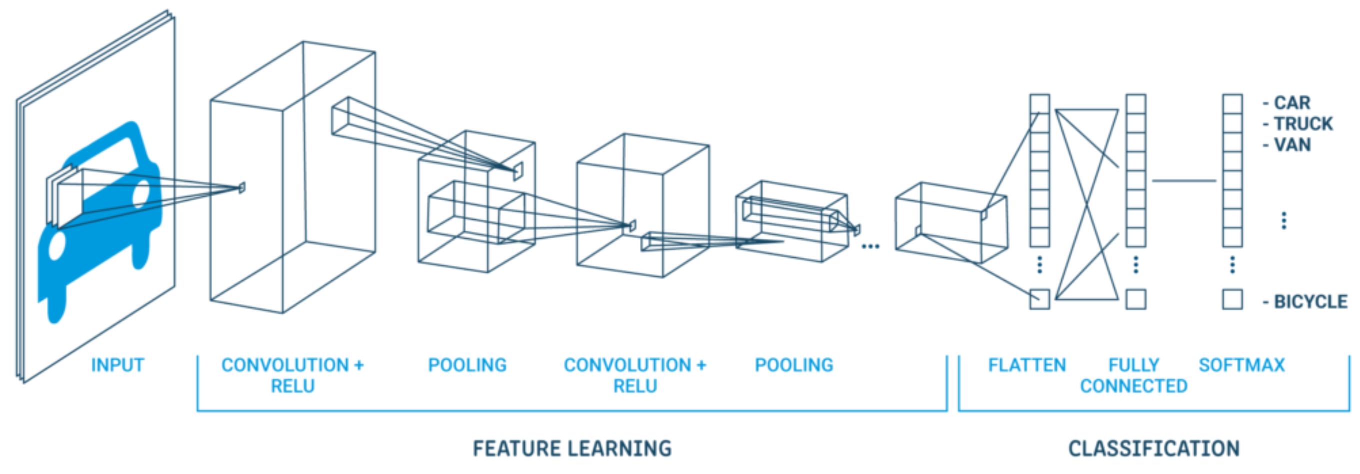
Step-3: Convert the data into float values

```
X_train=X_train.reshape(X_train.shape[0], 1,28,28).astype('float32') X_test
```

Step-4: Normalize the data

```
X_train=X_train/255 X_test=X_test/255 y_train= np_utils.to_categorical(y_tr
```

A classical CNN architecture looks like as shown below:



Output Layer (10 outputs)
Hidden Layer (128 neurons)
Flatten Layer
Dropout Layer 20%
Max Pooling Layer 2×2
Convolutional Layer 32 maps, 5×5
Visible Layer 1x28x28

The first hidden layer is a Convolutional layer call Convolution2D. It has 32 feature maps with size 5×5 and with a rectifier function. This is the input layer. Next is the pooling layer that takes the maximum value called MaxPooling2D. In this model, it is configured as a 2×2 pool size.

In the dropout layer regularization happens. It is set to randomly exclude 20% of the neurons in the layer to avoid overfitting. The fifth layer is the flattened layer that converts the 2D matrix data into a vector called Flatten. It allows the output to be fully processed by a standard fully connected layer.

Next, the fully connected layer with 128 neurons and rectifier activation function is used. Finally, the output layer has 10 neurons for the 10 classes and a softmax activation function to output probability-like predictions for each class.

Step-5: Run the model

```
def cnn_model():
    model=Sequential()
    model.add(Conv2D(32,5,5, padding='same',input_shape=(1,28,28), activation='relu')
    model.add(MaxPooling2D(pool_size=(2,2), padding='same'))
    model.add(Dropout(0.2))
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dense(num_classes, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accura
    return model

model=cnn_model()
model.fit(X_train, y_train, validation_data=(X_test,y_test),epochs=10, batch_size=200
score= model.evaluate(X_test, y_test, verbose=0)
print('The error is: %.2f%%'%(100-score[1]*100))
```

Output:

```
Epoch 1/10
300/300 - 2s - loss: 0.7825 - accuracy: 0.7637 - val_loss: 0.3071 - val_accuracy: 0.906
Epoch 2/10
300/300 - 1s - loss: 0.3505 - accuracy: 0.8908 - val_loss: 0.2192 - val_accuracy: 0.933
Epoch 3/10
300/300 - 1s - loss: 0.2768 - accuracy: 0.9126 - val_loss: 0.1771 - val_accuracy: 0.942
Epoch 4/10
300/300 - 1s - loss: 0.2392 - accuracy: 0.9251 - val_loss: 0.1508 - val_accuracy: 0.953
Epoch 5/10
300/300 - 1s - loss: 0.2164 - accuracy: 0.9325 - val_loss: 0.1423 - val_accuracy: 0.954
Epoch 6/10
300/300 - 1s - loss: 0.1997 - accuracy: 0.9380 - val_loss: 0.1279 - val_accuracy: 0.960
Epoch 7/10
```

```
300/300 - 1s - loss: 0.1856 - accuracy: 0.9415 - val_loss: 0.1179 - val_accuracy: 0.963
Epoch 8/10
300/300 - 1s - loss: 0.1777 - accuracy: 0.9433 - val_loss: 0.1119 - val_accuracy: 0.964
Epoch 9/10
300/300 - 1s - loss: 0.1689 - accuracy: 0.9469 - val_loss: 0.1093 - val_accuracy: 0.966
Epoch 10/10
300/300 - 1s - loss: 0.1605 - accuracy: 0.9493 - val_loss: 0.1053 - val_accuracy: 0.965
The error is: 3.41%
```

In the model results, it is visible as the number of epochs increases the accuracy improves. The error is 3.41%, lower the error higher the accuracy of the model.

How Convolutional Neural Networks Work?

Layers: CNNs consist of multiple layers including an input layer (for the image), convolutional layers, pooling layers, fully-connected layers, and an output layer (for classification).

Convolutional Layers play a crucial role in CNNs. Filters, like tiny squares, are utilized to scan across the image and detect characteristics such as edges, forms, and hues. Each filter is taught to recognize specific attributes.

Pooling Layers: These layers reduce the output resolution of convolutional layers, resulting in fewer parameters and less computational workload. Pooling includes compressing a portion of the image, like computing the mean or selecting the highest value.

In later stages, fully-connected layers operate akin to conventional neural networks by utilizing the complete output of the preceding layer for classification purposes.

Conclusion

Convolutional neural networks (CNNs) have emerged as powerful tools for image processing and computer vision tasks. This article provided a comprehensive guide to understanding CNNs, their architectural components, and their implementation using popular libraries like Keras and TensorFlow. Through hands-on examples with the MNIST dataset for handwritten digit recognition, readers gained practical experience in building, training, and evaluating CNN models for image classification problems. As deep learning continues to advance, CNNs will undoubtedly play a pivotal role in solving complex image processing challenges across various domains, from autonomous vehicles to medical imaging and beyond. With a solid grasp of CNN fundamentals, practitioners can confidently harness their potential for innovative applications.Hope you get understanding of CNN and image processing and how CNN algorithm for image processing.This tutorial is basically for CNN for beginners, that clarify everything about CNN and image preprocessing for cnn.

The media shown in this article on Image Processing using CNN are not owned by Analytics Vidhya and are used at the Author's discretion.

M Mohit

Advanced Algorithm Deep Learning Image Image Analysis Project Python Python Structured Data

★ 4.5

Getting Started with Large Language Models

Master Large Language Models (LLMs) with this course, offering clear guidance in NLP and model training made simple.

★ 4.6

Building LLM Applications using Prompt Engineering

This free course guides you on building LLM apps, mastering prompt engineering, and developing chatbots with enterprise data.

★ 4.8

Improving Real World RAG Systems: Key Challenges & Practical Solutions

Explore practical solutions, advanced retrieval strategies, and agentic RAG systems to improve context, relevance, and accuracy in AI-driven applications.

★ 4.7

Microsoft Excel: Formulas & Functions

Master MS Excel for data analysis with key formulas, functions, and LookUp tools in this comprehensive course.

RECOMMENDED ARTICLES

Frequently Asked Questions

Q1. What is CNN in image processing?

A. CNN stands for Convolutional Neural Network and is a type of deep learning algorithm used for analyzing and processing images. It performs a series of mathematical operations such as convolutions and pooling on an image to extract relevant features.

Q2. How is the working of CNN?

Q3. Why do we use CNN?