```
[ ] Start coding or generate with AI.
```

Introduction to CNN

Convolutional Neural Network is a Deep Learning algorithm specially designed for working with Images and videos. It takes images as inputs, extracts and learns the features of the image, and classifies them based on the learned features.

This algorithm is inspired by the working of a part of the human brain which is the Visual Cortex. The visual Cortex is a part of the human brain which is responsible for processing visual information from the outside world.

It has various layers and each layer has its own functioning i.e each layer extracts some information from the image or any visual and at last all the information received from each layer is combined and the image/visual is interpreted or classified.

Similarly, CNN has various filters, and each filter extracts some information from the image such as edges, different kinds of shapes (vertical, horizontal, round), and then all of these are combined to identify the image.

Now, the question here can be: Why can't we use Artificial Neural Networks for the same purpose? This is because there are some disadvantages with ANN:

It is too much computation for an ANN model to train large-size images and different types of image channels. The next disadvantage is that it is unable to capture all the information from an image whereas a CNN model can capture the spatial dependencies of the image. Another reason is that ANN is sensitive to the location of the object in the image i.e if the location or place of the same object changes, it will not be able to classify properly.

Components of CNN

The CNN model works in two steps: feature extraction and Classification

Feature Extraction is a phase where various filters and layers are applied to the images to extract the information and features out of it and once it's done it is passed on to the next phase i.e Classification where they are classified based on the target variable of the problem.

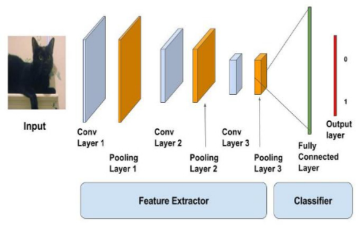
A typical CNN model looks like this:

Input layer

Convolution layer + Activation function

Pooling layer

Fully Connected Layer

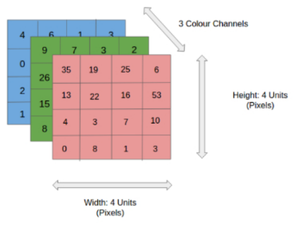


Double-click (or enter) to edit

Input layer

As the name says, it's our input image and can be Grayscale or RGB. Every image is made up of pixels that range from 0 to 255. We need to normalize them i.e convert the range between 0 to 1 before passing it to the model.

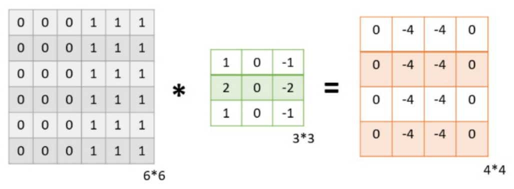
Below is the example of an input image of size 4*4 and has 3 channels i.e RGB and pixel values.



Double-click (or enter) to edit

Convolution Layer

The convolution layer is the layer where the filter is applied to our input image to extract or detect its features. A filter is applied to the image multiple times and creates a feature map which helps in classifying the input image. Let's understand this with the help of an example. For simplicity, we will take a 2D input image with normalized pixels.



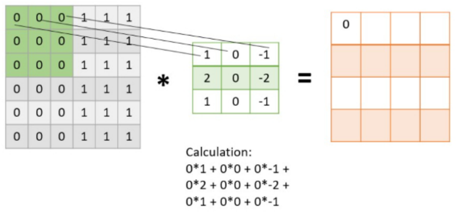
$$\underbrace{\begin{array}{|c|c|c|c|c|c|}\hline 0&0&0&1&1&1\\\hline 0&0&0&1&1&1\\\hline 0&0&0&1&1&1\\\hline 0&0&0&1&1&1\\\hline 0&0&0&1&1&1\\\hline 0&0&0&1&1&1\\\hline\end{array}}_{6*6} * \underbrace{\begin{array}{|c|c|c|}\hline 1&0&-1\\\hline 2&0&-2\\\hline 1&0&-1\\\hline\end{array}}_{3*3} = \underbrace{\begin{array}{|c|c|c|c|}\hline 0&-4&-4&0\\\hline 0&-4&-4&0\\\hline 0&-4&-4&0\\\hline 0&-4&-4&0\\\hline\end{array}}_{4*4}$$

In the above figure, we have an input image of size 66 *and applied a filter of* 33 on it to detect some features.

In this example, we have applied only one filter but in practice, many such filters are applied to extract information from the image.

The result of applying the filter to the image is that we get a Feature Map of 4*4 which has some information about the input image. Many such feature maps are generated in practical applications.

Let's get into some maths behind getting the feature map in the above image.



Double-click (or enter) to edit

As presented in the above figure, in the first step the filter is applied to the green highlighted part of the image, and the pixel values of the image are multiplied with the values of the filter (as shown in the figure using lines) and then summed up to get the final value.

In the next step, the filter is shifted by one column as shown in the below figure. This jump to the next column or row is known as stride and in this example, we are taking a stride of 1 which means we are shifting by one column.

link text

Double-click (or enter) to edit

Similarly, the filter passes over the entire image and we get our final Feature Map. Once we get the feature map, an activation function is applied to it for introducing nonlinearity.

A point to note here is that the Feature map we get is smaller than the size of our image. As we increase the value of stride the size of the feature map decreases.

Double-click (or enter) to edit

1. importing the required libraries

```
#importing the required libraries
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPool2D
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Dense
```

2. loading data

```
(X_train,y_train) , (X_test,y_test)=mnist.load_data()
```

3. reshaping data

```
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], X_train.shape[2], 1))
X_test = X_test.reshape((X_test.shape[0],X_test.shape[1],X_test.shape[2],1))
```

4.checking the shape after reshaping

```
print(X_train.shape)
print(X_test.shape)
```

5. normalizing the pixel values

```
X_train=X_train/255
X_test=X_test/255
```

6. defining model

```
model=Sequential()model=Sequential()
```

7. adding convolution layer

```
model.add(Conv2D(32,(3,3),activation='relu',input_shape=(28,28,1)))
```

8. adding pooling layer

```
model.add(MaxPool2D(2,2))
```

9. adding fully connected layer

```
model.add(Flatten())
model.add(Dense(100,activation='relu'))
```

10. adding output layer

```
model.add(Dense(10,activation='softmax'))
```

11. compiling the model

```
model.compile(loss='sparse_categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
```

Double-click (or enter) to edit

12. fitting the model

```
model.fit(X_train,y_train,epochs=10)
```

Output:

```
Epoch 1/10
1875/1875 [==============================] - 27s 14ms/step - loss: 0.8464 - accuracy: 0.7492
Epoch 2/10
1875/1875 [==============================] - 25s 13ms/step - loss: 0.3448 - accuracy: 0.8985
Epoch 3/10
1875/1875 [==============================] - 18s 10ms/step - loss: 0.2882 - accuracy: 0.9149
Epoch 4/10
1875/1875 [==============================] - 18s 9ms/step - loss: 0.2433 - accuracy: 0.9281
Epoch 5/10
1875/1875 [==============================] - 18s 10ms/step - loss: 0.2081 - accuracy: 0.9383
```

```
Epoch 6/10
1875/1875 [==============================] - 18s 10ms/step - loss: 0.1841 - accuracy: 0.9442
Epoch 7/10
1875/1875 [==============================] - 18s 10ms/step - loss: 0.1670 - accuracy: 0.9502
Epoch 8/10
1875/1875 [==============================] - 18s 9ms/step - loss: 0.1532 - accuracy: 0.9546
Epoch 9/10
1875/1875 [==============================] - 17s 9ms/step - loss: 0.1426 - accuracy: 0.9578
Epoch 10/10
1875/1875 [==============================] - 18s 10ms/step - loss: 0.1329 - accuracy: 0.9600
```

13. evaluting the model

```
model.evaluate(X_test,y_test)
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.1399 - accuracy: 0.9587
[0.13990521430969238, 0.9587000012397766]
```

Double-click (or enter) to edit