Convolutional neural network 2D with module tf.nn

This project aims at realizing a model based on 2D convolutional neural networks using the tf.nn module in TensorFlow. The model has the ability to classify eight items ['airplane', 'car', 'cat', 'dog', 'flower', 'fruit', 'motorbike', 'person'] and has been trained with 6899 images. The description of this project is mainly focused on describing the functionality of 2D convolutional neural networks.

Convolutional neural network 2D whit module tf.nn

Alin - Cosmin TOT

Elementary functions in the structure of CNN models

In most cases, CNN models contain at least three important elements, described below.

- CNN
- Pool
- Activate function

Convolutional neural networks

Convolutional Neural Networks (CNNs) are an advanced version of Artificial Neural Networks (ANNs), mainly designed to extract features from matrix datasets. These types of networks are used primarily to deal with video, audio, and text content databases. As illustrated in Figure 1.1, the components of a simple 2-dimensional CNN are: input data matrix, filter, and map.

The feature detector, i.e., the filter, is a two-dimensional matrix of weights that calculates over the entire area of the matrix containing the input data. The pitch of the filter must be defined on both horizontal and vertical axes. If the filter step is greater than 1, the size of the output matrix Map will be smaller than the input matrix. The elements of the Map matrix are the result of the weighted sum of the filter matrix and the area of the input matrix where the filter maps. So, each step of the filter results in one element in the Map matrix. An important component to note is that the value of the weights in the filter matrix remains constant throughout the time that the filter is applied to the area of the input matrix.

In the case of a 3-channel (RGB) image, we have a 3-dimensional matrix as input data, and the approach is similar to the previous description. Figure 1.2 illustrates the workflow of a CNN with a 3-dimensional (RGB) matrix as input and a 3-dimensional matrix as output (i.e., n Map matrices). In this case, n filters are required, corresponding to n outputs. Each filter consists of 3 2D matrices that are culled on the corresponding matrix in the RGB matrix, resulting in 3 matrices (YR, YG, YB) where each element is the weighted sum of the filter displacements. Each 'n' Map matrix is the result of the sum of the corresponding (n) matrices YR, YG, and YB. Equation 1 presents in analytic form the above description, and one can observe the functionality much more precisely.

Fig. 1.1 One core CNN with 2D data input and one map output

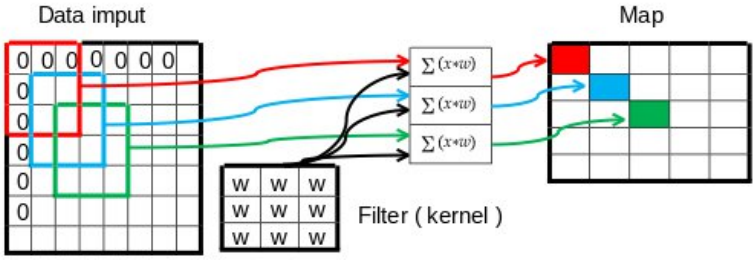
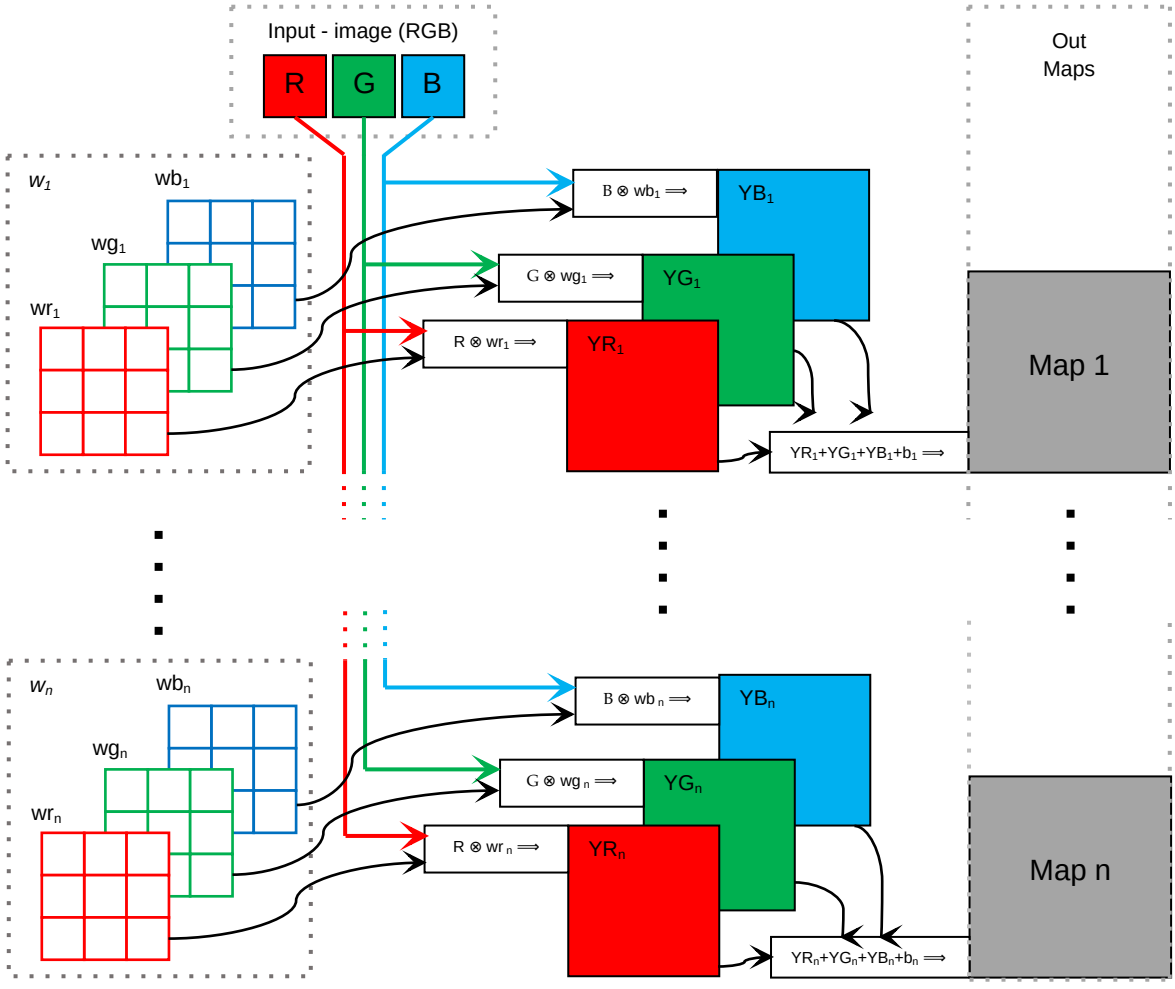


Fig. 1.2 One core CNN with 3D data input and n maps output



Convolutional neural network 2D whit module tf.nn

Alin - Cosmin TOT

With the tf.nn.conv2d() method, we can compute a 2D convolution using a matrix with 4D input = [batch, height, width, channels] as input and a filter (tensor). In general, the filter is realized as a matrix with random elements, with the form filter = [filter_height, filter_width, no. in_channels, no. out_channels]. If data_format = 'NHWC' the filter displacement will be strides = [batch, in_height, in_width, in_channels]. The mode of the displacement on the edge of the matrix is assigned as padding= 'VALID' or 'SAME', i.e. the filter edge not to exceed the matrix or the center of the filter to reach the edge of the metric on which it is culled, adding 0's on the edge of the input matrix. Method expression: tf.nn.conv2d(input, filter, strides, padding, data_format=None, name=None)

(1)

$$\begin{bmatrix} 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & R_{11} & R_{12} & R_{13} & \cdots & 0 \\ 0 & R_{21} & R_{22} & R_{23} & \cdots & 0 \\ 0 & R_{31} & R_{32} & R_{33} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \otimes \boxed{\begin{matrix} wr_{1.11} & wr_{1.12} & wr_{1.13} \\ wr_{1.21} & wr_{1.22} & wr_{1.23} \\ wr_{1.31} & wr_{1.32} & wr_{1.33} \end{matrix}} + \begin{bmatrix} 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & G_{11} & G_{12} & G_{13} & \cdots & 0 \\ 0 & G_{21} & G_{22} & G_{23} & \cdots & 0 \\ 0 & G_{31} & G_{32} & G_{33} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \otimes \boxed{\begin{matrix} wg_{1.11} & wg_{1.12} & wg_{1.13} \\ wg_{1.21} & wg_{1.22} & wg_{1.23} \\ wg_{1.31} & wg_{1.32} & wg_{1.33} \end{matrix}} + \begin{bmatrix} 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & B_{11} & B_{12} & B_{13} & \cdots & 0 \\ 0 & B_{21} & B_{22} & B_{23} & \cdots & 0 \\ 0 & B_{31} & B_{32} & B_{33} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \otimes \boxed{\begin{matrix} wb_{1.11} & wb_{1.12} & wb_{1.13} \\ wb_{1.21} & wb_{1.22} & wb_{1.23} \\ wb_{1.31} & wb_{1.32} & wb_{1.33} \end{matrix}} + b_1$$

$$\vdots$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & R_{11} & R_{12} & R_{13} & \cdots & 0 \\ 0 & R_{21} & R_{22} & R_{23} & \cdots & 0 \\ 0 & R_{31} & R_{32} & R_{33} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \otimes \boxed{\begin{matrix} wr_{n.11} & wr_{n.12} & wr_{n.13} \\ wr_{n.21} & wr_{n.22} & wr_{n.23} \\ wr_{n.31} & wr_{n.32} & wr_{n.33} \end{matrix}} + \begin{bmatrix} 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & G_{11} & G_{12} & G_{13} & \cdots & 0 \\ 0 & G_{21} & G_{22} & G_{23} & \cdots & 0 \\ 0 & G_{31} & G_{32} & G_{33} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \otimes \boxed{\begin{matrix} wg_{n.11} & wg_{n.12} & wg_{n.13} \\ wg_{n.21} & wg_{n.22} & wg_{n.23} \\ wg_{n.31} & wg_{n.32} & wg_{n.33} \end{matrix}} + \begin{bmatrix} 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & B_{11} & B_{12} & B_{13} & \cdots & 0 \\ 0 & B_{21} & B_{22} & B_{23} & \cdots & 0 \\ 0 & B_{31} & B_{32} & B_{33} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \otimes \boxed{\begin{matrix} wb_{n.11} & wb_{n.12} & wb_{n.13} \\ wb_{n.21} & wb_{n.22} & wb_{n.23} \\ wb_{n.31} & wb_{n.32} & wb_{n.33} \end{matrix}} + b_n$$

$$YR_{n.11} = 0 \cdot rw_{n.11} + 0 \cdot rw_{n.12} + 0 \cdot rw_{n.13} + 0 \cdot rw_{n.21} + R_{11} \cdot rw_{n.22} + R_{12} \cdot rw_{n.23} + 0 \cdot rw_{n.31} + R_{21} \cdot rw_{n.32} + R_{22} \cdot rw_{n.33}$$

$$YR_{n.12} = 0 \cdot rw_{n.11} + 0 \cdot rw_{n.12} + 0 \cdot rw_{n.13} + R_{11} \cdot rw_{n.21} + R_{12} \cdot rw_{n.22} + R_{13} \cdot rw_{n.23} + R_{21} \cdot rw_{n.31} + R_{22} \cdot rw_{n.32} + R_{23} \cdot rw_{n.33}$$

$$\vdots$$

$$YR_{n.21} = 0 \cdot rw_{n.11} + R_{11} \cdot rw_{n.12} + R_{12} \cdot rw_{n.13} + 0 \cdot rw_{n.21} + R_{21} \cdot rw_{n.22} + R_{22} \cdot rw_{n.23} + 0 \cdot rw_{n.31} + R_{31} \cdot rw_{n.32} + R_{32} \cdot rw_{n.33}$$

$$YR_{n.22} = R_{11} \cdot rw_{n.11} + R_{12} \cdot rw_{n.12} + R_{13} \cdot rw_{n.13} + R_{21} \cdot rw_{n.21} + R_{22} \cdot rw_{n.22} + R_{23} \cdot rw_{n.23} + R_{31} \cdot rw_{n.31} + R_{32} \cdot rw_{n.32} + R_{33} \cdot rw_{n.33}$$

$$\vdots$$

$$YG_{n.11} = 0 \cdot gw_{n.11} + 0 \cdot gw_{n.12} + 0 \cdot gw_{n.13} + 0 \cdot gw_{n.21} + G_{11} \cdot gw_{n.22} + G_{12} \cdot gw_{n.23} + 0 \cdot gw_{n.31} + G_{21} \cdot gw_{n.32} + G_{22} \cdot gw_{n.33}$$

$$YG_{n.12} = 0 \cdot gw_{n.11} + 0 \cdot gw_{n.12} + 0 \cdot gw_{n.13} + G_{11} \cdot gw_{n.21} + G_{12} \cdot gw_{n.22} + G_{13} \cdot gw_{n.23} + G_{21} \cdot gw_{n.31} + G_{22} \cdot gw_{n.32} + G_{23} \cdot gw_{n.33}$$

$$\vdots$$

$$YG_{n.21} = 0 \cdot gw_{n.11} + G_{11} \cdot gw_{n.12} + G_{12} \cdot gw_{n.13} + 0 \cdot gw_{n.21} + G_{21} \cdot gw_{n.22} + G_{22} \cdot gw_{n.23} + 0 \cdot gw_{n.31} + G_{31} \cdot gw_{n.32} + G_{32} \cdot gw_{n.33}$$

$$YG_{n.22} = G_{11} \cdot gw_{n.11} + G_{12} \cdot gw_{n.12} + G_{13} \cdot gw_{n.13} + G_{21} \cdot gw_{n.21} + G_{22} \cdot gw_{n.22} + G_{23} \cdot gw_{n.23} + G_{31} \cdot gw_{n.31} + G_{32} \cdot gw_{n.32} + G_{33} \cdot gw_{n.33}$$

$$\vdots$$

Convolutional neural network 2D whit module tf.nn

Alin - Cosmin TOT

$$YB_{n.11} = 0 \cdot bw_{n.11} + 0 \cdot bw_{n.12} + 0 \cdot bw_{n.13} + 0 \cdot bw_{n.21} + B_{11} \cdot bw_{n.22} + B_{12} \cdot bw_{n.23} + 0 \cdot bw_{n.31} + B_{21} \cdot bw_{n.32} + B_{22} \cdot bw_{n.33}$$

$$YB_{n.12} = 0 \cdot bw_{n.11} + 0 \cdot bw_{n.12} + 0 \cdot bw_{n.13} + B_{11} \cdot bw_{n.21} + B_{12} \cdot bw_{n.22} + B_{13} \cdot bw_{n.23} + B_{21} \cdot bw_{n.31} + B_{22} \cdot bw_{n.32} + B_{23} \cdot gw_{n.33}$$

$$\vdots$$

$$YB_{n.21} = 0 \cdot bw_{n.11} + B_{11} \cdot bw_{n.12} + B_{12} \cdot gw_{n.13} + 0 \cdot bw_{n.21} + B_{21} \cdot bw_{n.22} + B_{22} \cdot bw_{n.23} + 0 \cdot bw_{n.31} + B_{31} \cdot bw_{n.32} + B_{32} \cdot bw_{n.33}$$

$$YB_{n.22} = B_{11} \cdot bw_{n.11} + B_{12} \cdot bw_{n.12} + B_{13} \cdot bw_{n.13} + B_{21} \cdot bw_{n.21} + B_{22} \cdot bw_{n.22} + B_{23} \cdot bw_{n.23} + B_{31} \cdot bw_{n.31} + B_{32} \cdot bw_{n.32} + B_{33} \cdot bw_{n.33}$$

$$\vdots$$

The result has the following form:

$$\boldsymbol{Map_1} = \begin{bmatrix} YR_{1.11} + YG_{1.11} + YB_{1.11} + b_1 & YR_{1.12} + YG_{1.12} + YB_{1.12} + b_1 & \dots \\ YR_{1.21} + YG_{1.21} + YB_{1.21} + b_1 & YR_{1.22} + YG_{1.22} + YB_{1.22} + b_1 & \dots \\ \vdots & \vdots & \ddots \end{bmatrix}$$

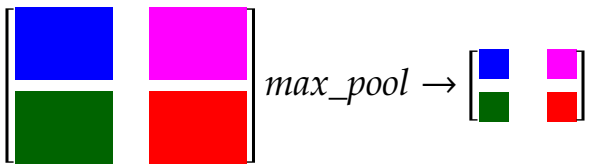
$$\vdots$$

$$\boldsymbol{Map_n} = \begin{bmatrix} YR_{n.11} + YG_{n.11} + YB_{n.11} + b_n & YR_{n.12} + YG_{n.12} + YB_{n.12} + b_n & \dots \\ YR_{n.21} + YG_{n.21} + YB_{n.21} + b_n & YR_{n.22} + YG_{n.22} + YB_{n.22} + b_n & \dots \\ \vdots & \vdots & \ddots \end{bmatrix}$$

Pool

The pooling layer is used in the structure of CNNs in order to reduce the size of feature maps while preserving the relevant information. The functionality of this method (Fig. 2.1) is based, as before, on a two-dimensional filter that slides on the feature map channel.

Fig. 2.1



For example, the max_plot method takes the maximum value in the area it maps to. Setting the functionality of the tf.nn.max_pool() method involves specifying the window shape, ksize=[corresponding batch no, filter height, filter width, channel number], and the filter displacement strides, [corresponding batch no, filter height, filter width, channel number]. Padding of the filter on the edge of the feature matrix padding= 'VALID' or 'SAME', i.e. the edge of the filter not to exceed the matrix or the center of the filter to reach the edge of the metric on which it pads, adding 0 - ions on the edge of the input matrix.

Activate function

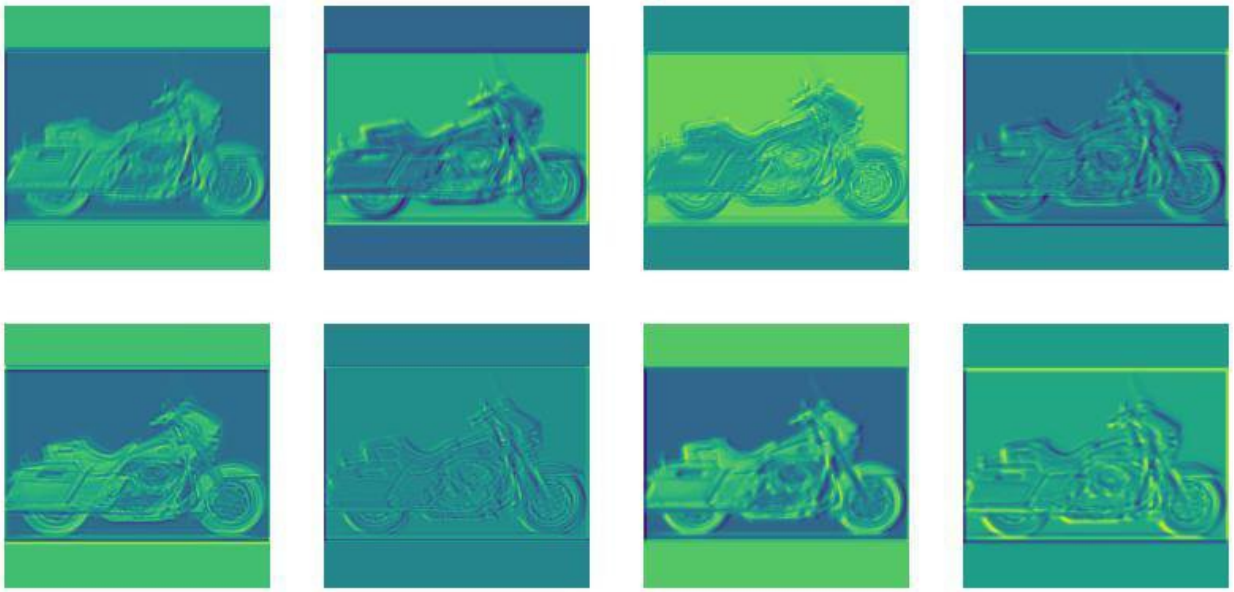
The activation function is meant to transform the linear form of the weighted sum to a nonlinear form, i.e., the output does not change proportionally to the input. In most real-world situations, the data have complex shapes, and by adding a nonlinear function such as ReLU, Sigmoid, or Tanh, the network can create curved decision boundaries to separate them correctly. A commonly used function is the function ReLU, with the relation f(x)=max(0,x).

Influența CNN, Pool și Activate function, asupra imagini

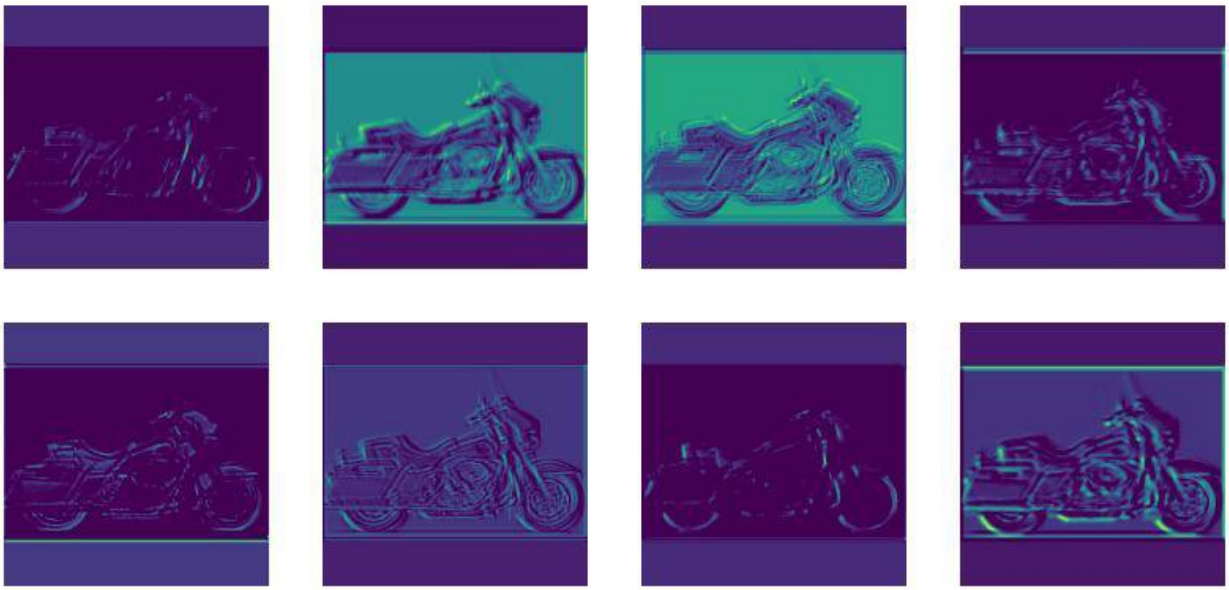
In the following, a series of images, taken from the attached program, are presented, and the recursion of image modifications can be observed.

original image (224X224)

convolutions 1 (224, 224, 8)



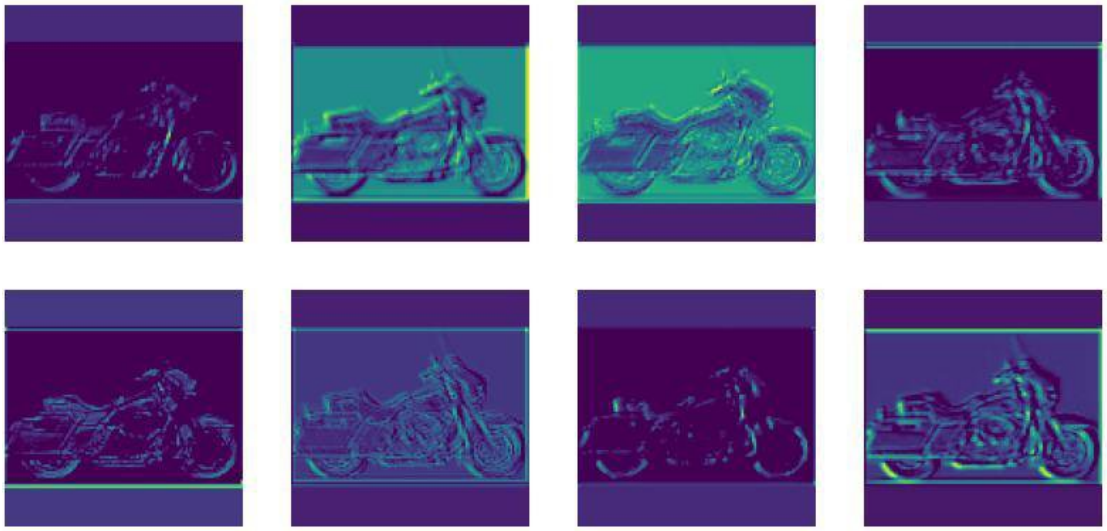
activate 1 (224, 224, 8)



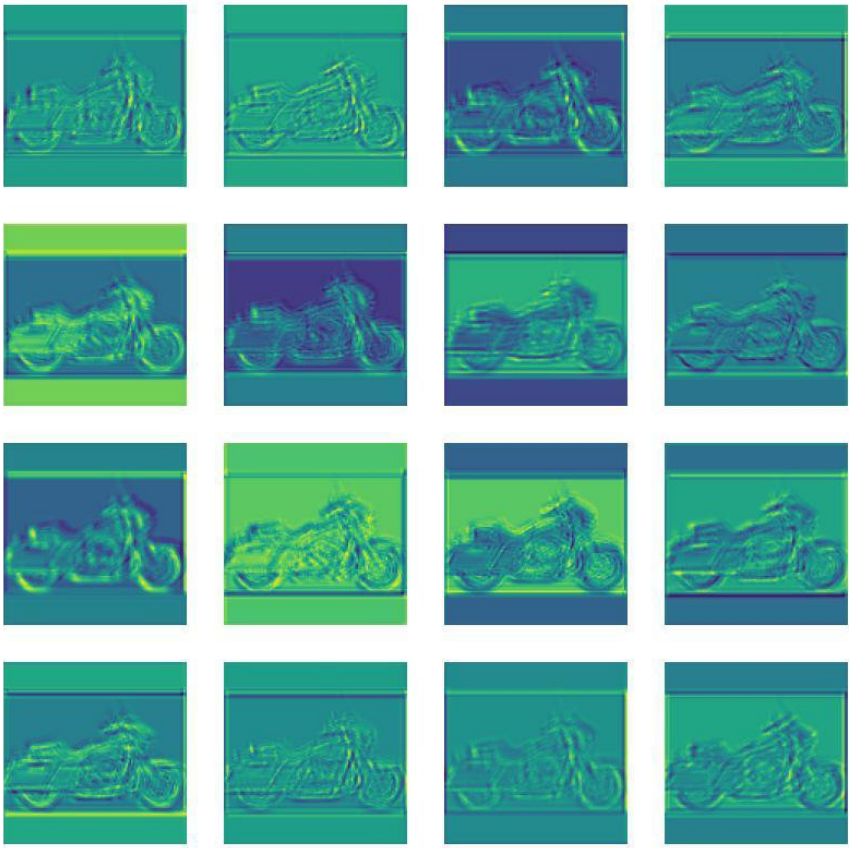
Convolutional neural network 2D whit module tf.nn

Alin - Cosmin TOT

pool 1 (112, 112, 8)



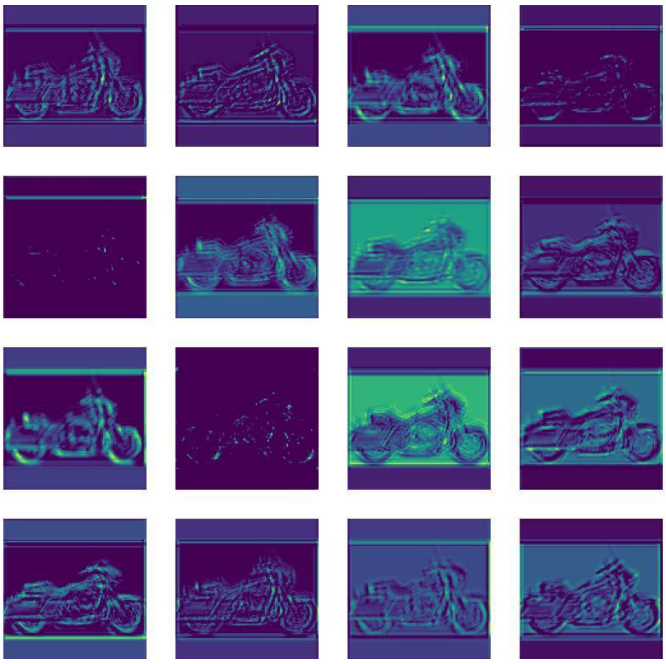
convolutions 2 (112, 112, 16)



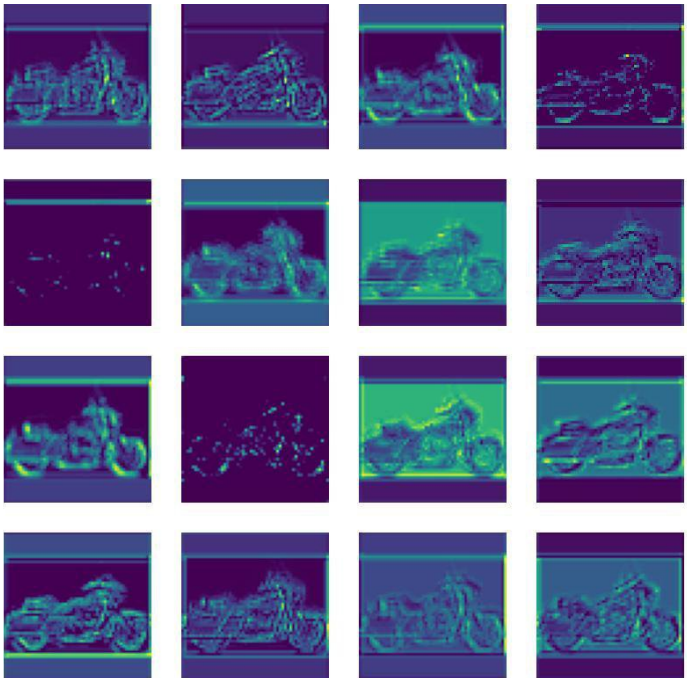
Convolutional neural network 2D whit module tf.nn

Alin - Cosmin TOT

activate 2 (112, 112, 16)



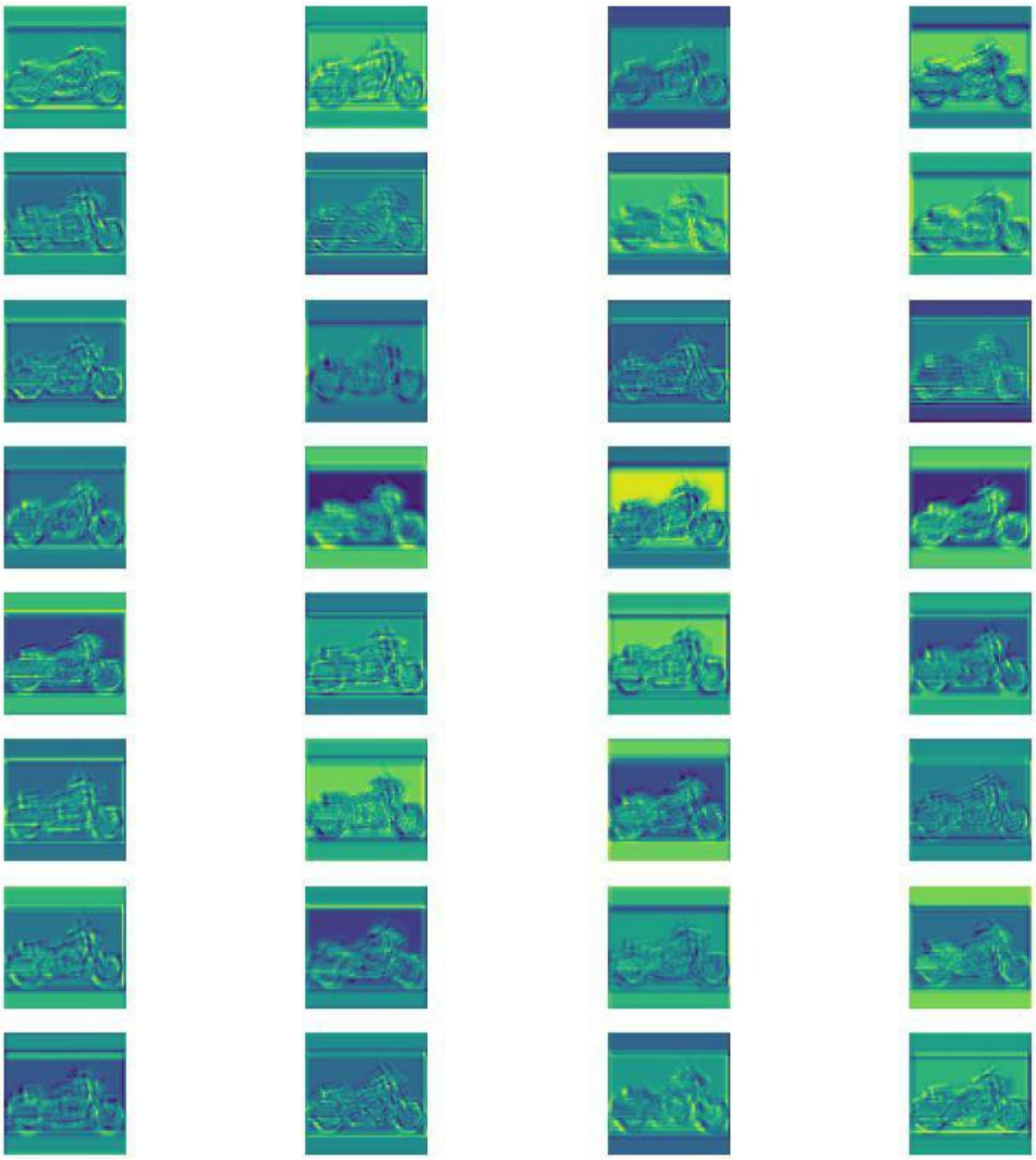
pool 2 (56, 56, 16)



Convolutional neural network 2D whit module tf.nn

Alin - Cosmin TOT

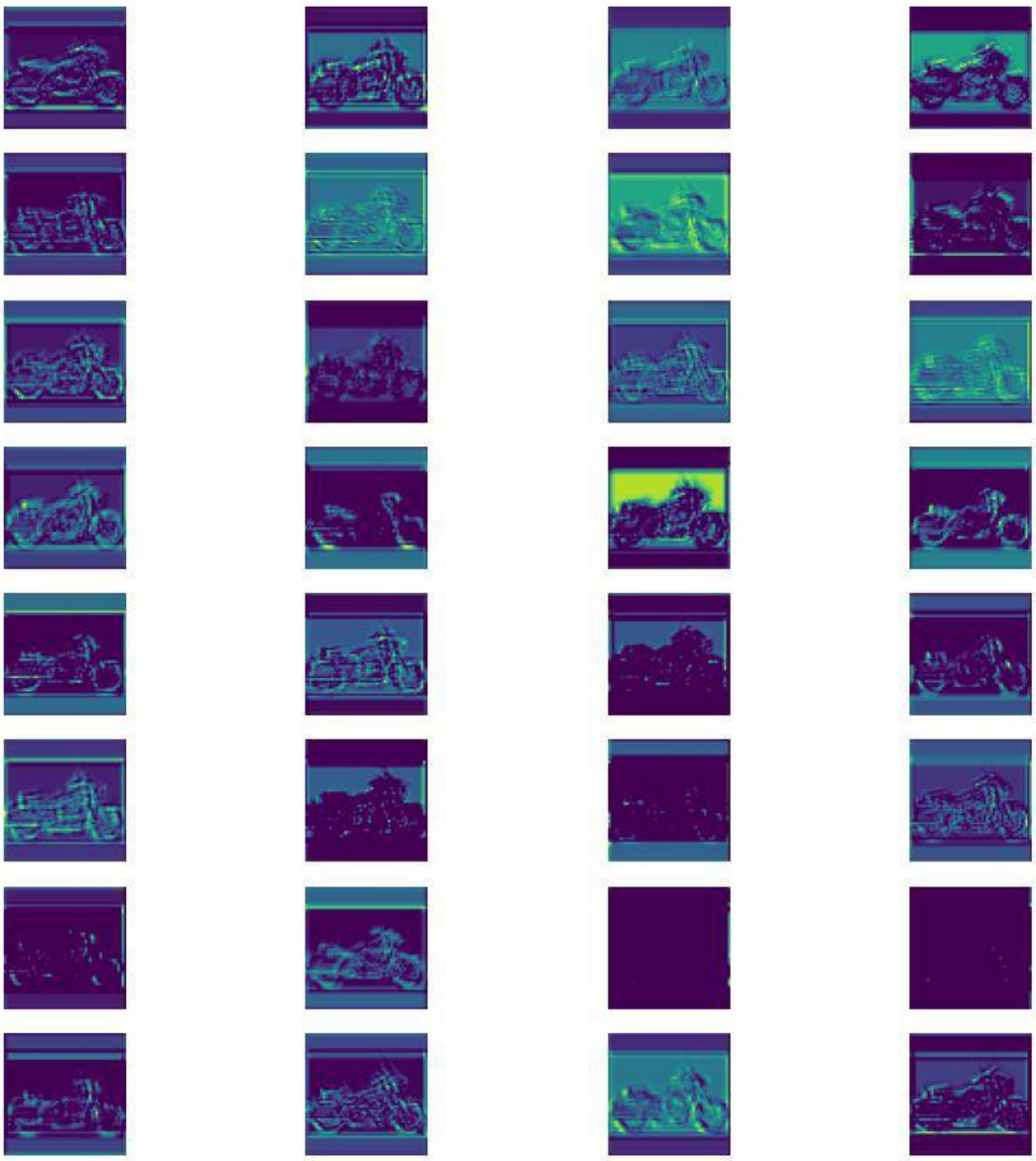
convolutions 3 (56, 56, 32)



Convolutional neural network 2D whit module tf.nn

Alin - Cosmin TOT

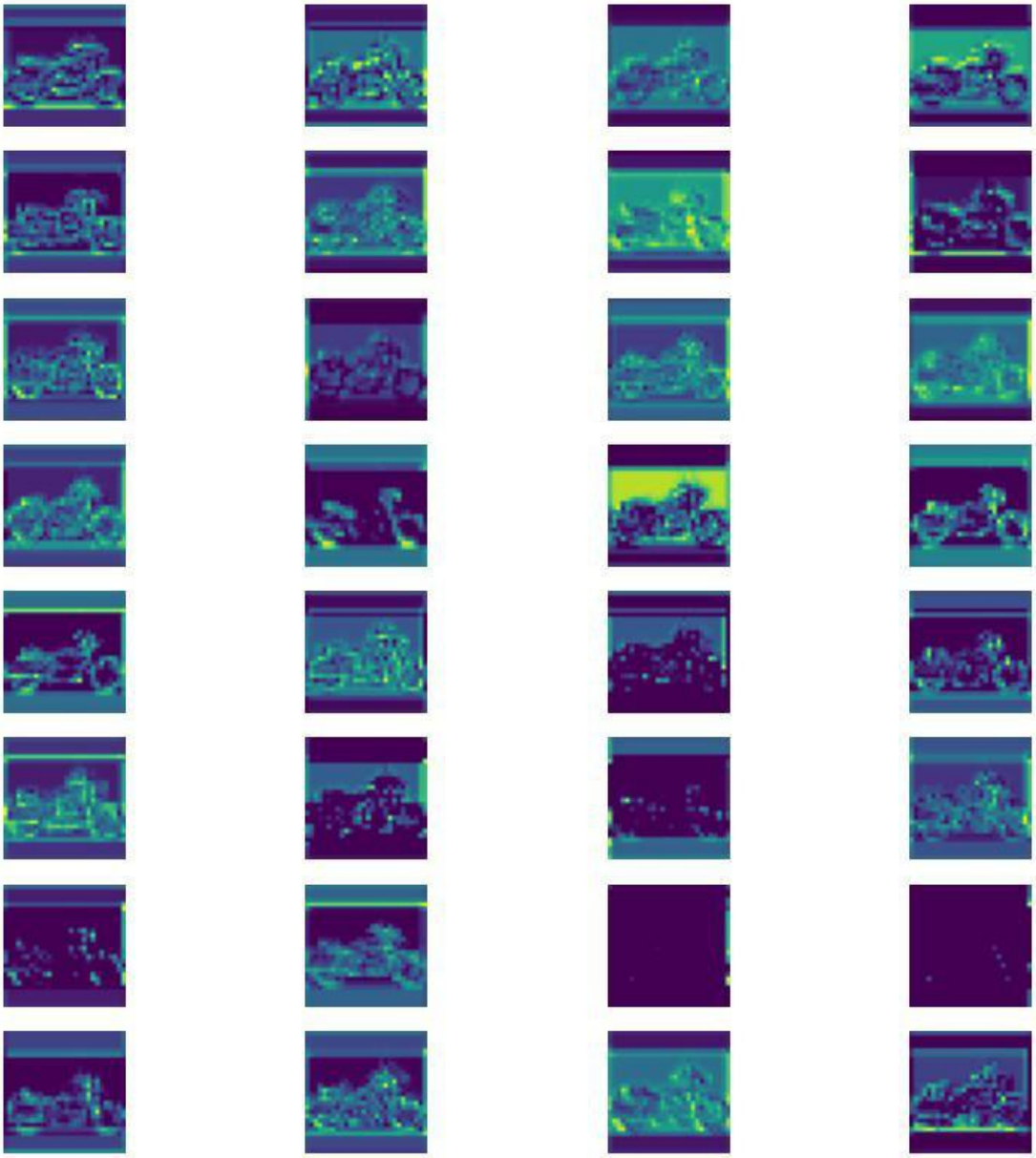
activate 3 (56, 56, 32)



Convolutional neural network 2D whit module tf.nn

Alin - Cosmin TOT

pool 3 (28, 28, 32)



Convolutional neural network 2D whit module tf.nn

Alin - Cosmin TOT

Bibliografie :

https://www.datacamp.com/tutorial/introduction-to-convolutional-neural-networks-cnns

https://www.simplilearn.com/tutorials/deep-learning-tutorial/convolutional-neural-network

https://www.geeksforgeeks.org/deep-learning/relu-activation-function-in-deep-learning/

https://www.datacamp.com/blog/rectified-linear-unit-relu

https://www.geeksforgeeks.org/deep-learning/cnn-introduction-to-pooling-layer/

https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/