UNIWERSYTET TECHNOLOGICZNO-PRZYRODNICZY
IM. JANA I JĘDRZEJA ŚNIADECKICH W BYDGOSZCZY
ZESZYTY NAUKOWE NR 270
TELEKOMUNIKACJA I ELEKTRONIKA 23 (2019) 5–21

APPLICATION OF CONVOLUTIONAL NEURON NETWORK FOR IMAGE PROCESSING AND INTERPRETATION

Krzysztof Pałczyński

Faculty of Telecommunications, Computer Science and Electrical Engineering,
UTP University of Science and Technology, Al. prof. S. Kaliskiego 7, 85-796 Bydgoszcz
e-mail: krzysztof@palczynski.com

Summary: This article describes the application of Convolutional Neural Network in image processing and describes how it works. There are presented: network layers, types of activation functions, example of the AlexNet network architecture, the use of the loss function and the cross entropy method to calculate the loss during tests, L2 and Dropout methods used for weights regularization and optimization of the loss function using Stochastic Gradient Drop.

Keywords: Convolutional Neural Network, image processing, filtration, convolution, activation function, loss function, softmax, cross entropy, L2, Dropout, Stochastic Gradient Drop

1. INTRODUCTION

Image processing is an important scope of research in the modern world. The computer's ability to understand what a given graphics describes and the use of this knowledge is needed in such areas as identification of a person, diagnosis of a disease, description of a photo in natural language or data compression. However, it is a difficult task due to the following factors:

- Input data size – raster graphics coded in the RGB system are three-dimensional tensors, in which the first two dimensions are responsible for the length and width of the image, and the third dimension describes three color channels. The first two dimensions contain a significant amount of data to properly represent information. For example, a relatively small 224 by 224 pixel graphics contains $224 * 224 * 3 = 150\,528$ data samples. Despite the small size, and hence inferior image quality, the input tensor is large and requires many computations to obtain satisfactory data from it.
- A multitude of ways to represent the same phenomenon – photos depicting the same object can be taken from a different angle, with diverse illuminations and from various distances. In addition, photos are susceptible to noise resulting from equipment inaccuracies and quantization errors. This means that the algorithm for interpreting images must be able to recognize the same object even though every graphics depicting the same phenomenon may have distinct set of pixels.
- Graphics ambiguity – the image can present object of class A while being similar to a typical representation of class B objects. The graphics classification algorithm must be able to distinguish between these classes despite the lack of a deterministic way of differentiating between objects.

Convolutional Neural Network is a new approach to solve described problem using machine learning. It is suitable for interpreting graphics, which is confirmed, among others, by the international Image Net Large Scale Visual Recognition Challenge (ILSVRC for short). In this contest, the ability of the algorithms to accurately recognize and classify the objects shown in the graphics is checked. Since 2012, the winners of this competition are convolution neural networks. Due to its successes, this branch of machine learning is gaining popularity in both academic and industrial circles. The advantages of convolutional neural networks include the small amount of pre-processing data needed, the ability to interpret photos despite differences in illumination, viewing angle and image size, reduced chance of over-matching to test data, and a shorter learning process compared to typical neural networks.

2. CONVOLUTIONAL NEURON NETWORK (CNN) – BASIC

Convolutional neural network is a type of deep neural networks with a hierarchical structure. There is a fundamental difference between fully connected layers (*FC* for short) and convolutional layers. In FC layer, the outputs of all neurons inside N-1 layer are connected to the inputs of all N-layer neurons, while in the convolutional layer, the output of neuron inside N layer is connected to a small number of N-1-layer neurons. This distinction is shown in the Fig. 1.

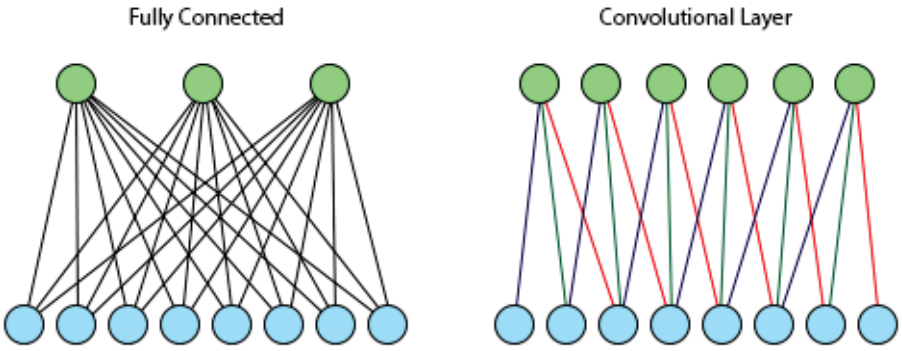


Fig. 1. Graphic comparison of the Fully Connected layer to the Convolutional Layer

This structure of the convolutional neural network is dictated by the need for regularization. Networks consisting only of FC layers tend to over-fit with training data resulting in problems in interpreting information received from outside the learning period. The problem is presented in the Fig. 2.

Blue points are data received for input, while the black lines are approximations made by the neural network. On the left there is the desired adjustment of the neural network to the data presented, while on the right is excessive mapping. In the first case, the network accurately approximated the value of only a few points, but created a simple data model represented in this case by a parabolic curve. The mapping function has only one monotonicity change point and a high tolerance factor for abnormal measurements. In contrast, the over-matching on the right results in a very complicated data model, which by its complexity is susceptible to anomalous information samples and is generally unstable. This data model very well replicates the examples used for neural network training, but it is not appropriate to interpret data in the same semantic range that the network could not learn.

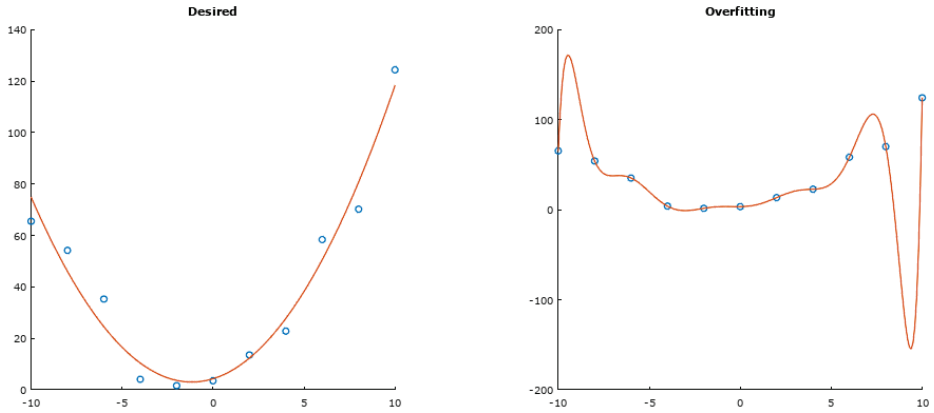


Fig. 2. Graphical comparison of the desired function approximation to the over-matching function

The convolutional neural network is not a homogeneous network. It consists of different types of layers stacked on top of each other. It consists of layers: convolutional, rectifier, pooling and fully connected.

3. CONVOLUTIONAL LAYER

The convolutional layer extracts features of the introduced image. Each convolutional layer has its own filters (kernels) responsible for reducing the input tensor to the form of an activation map of diminished size. In this way, the neural network carries out the process of abstracting the input graphics. Each convolutional layer filters the input data consistently reducing their size and passing the map of extracted features to the next layer. The neural network learns by correcting the filters.

The result of the graphic processing by the convolutional layer is the activation map. The activation map is a three-dimensional tensor, in which the first and second dimensions represent the result of the input image filtration, and the third dimension is the combination of subsequent image processing results by the convolutional layer filters. The convolutional layer filter tensor is a square matrix. Image processing involves making a filter convolute through subsequent areas of graphics corresponding in size and this process can be described by the following formula (1):

$$Y_{i,j} = \sum_{c=1}^{a_l} \sum_{d=1}^{a_w} \left(a_{c,d} x_{(i+c-1),(j+d-1)} \right) \tag{1}$$

where:

$Y_{i,j}$ – the resulting tensor value in the i-th row and j-th column,
$x_{i,j}$ – input tensor value in the i-th row and j-th column,
$a_{i,j}$ – value of the input filter in the i-th row and j-th column,
a_l, a_w – filter length and width.

This process is shown in the (Fig. 3) below:

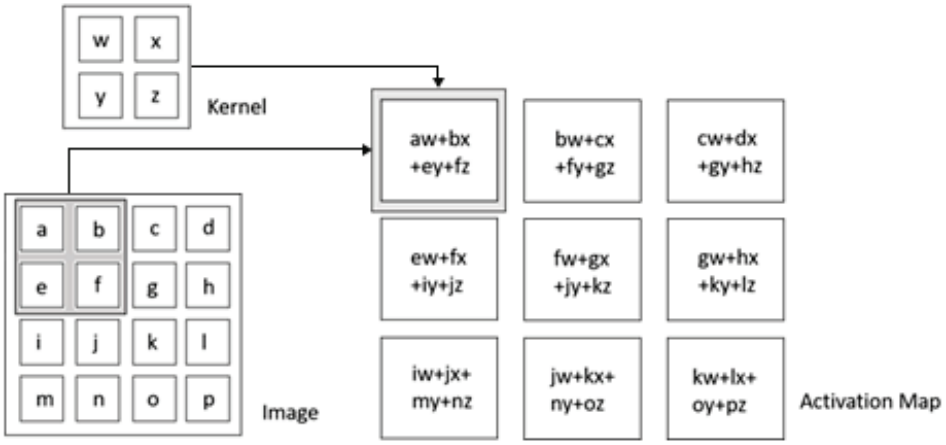


Fig. 3. Presentation of the process of applying the filter to the input image inside convolutional layer

In the case shown in the Figure (Fig. 3), the image is a 4x4 tensor, while the filter is a 2x2 square matrix. The 4x4 square matrix can be divided into 9 overlapping 2x2 areas, so the activation map is 3x3. Each numerical value contained in the activation map is the result of convolution of the image region matrix by the filter.

As the image passes through subsequent convolutional layers, its length and width are progressively reduced while its depth may increase or decrease depending on the number of filters used and the network architecture. With each subsequent step, the graphics are more and more abstracted, less and less reminiscent of the original input tensor, consisting of increasingly clearly extracted features that allow image interpretation.

The convolutional layer has optional parameters such as the stride and the padding. The stride informs the convolutional layer what is the distance between successive areas of aggregation of input values. In the case presented above, the stride is equal to 1, which means that the window calculating the value of subsequent cells of the result matrix is moved by 1. The Figure below (Fig. 3) shows the case in which the stride is equal to 2.

The window calculating the value of a cell with coordinates [0; 1] is shifted by 2 units horizontally relative to the area calculating the value for the cell [0; 0]. However, the window for cell [1; 0] is shifted by 2 units vertically relative to the area responsible for cell [0; 0]. The higher the stride value, the smaller the size of the resulting map. This parameter is useful in the first convolutional layers to quickly reduce the number of results that contribute little to the process of image analysis and interpretation.

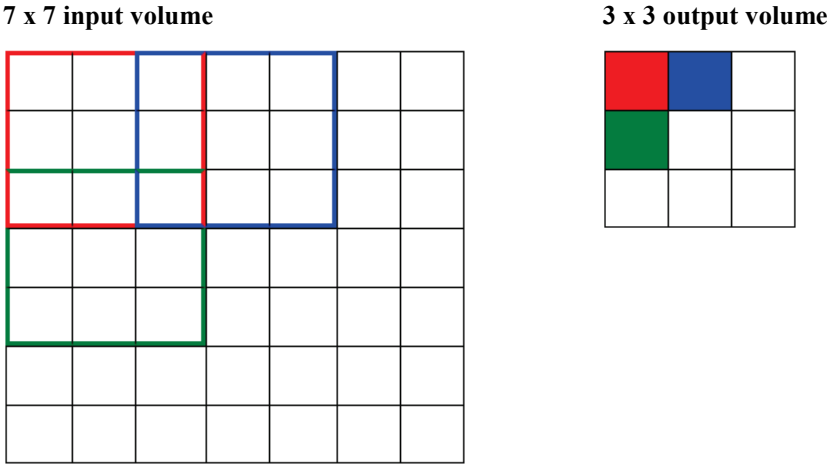


Fig. 4. Graphic representation of the impact of the stride parameter on the filtration process

The padding is a parameter by which the input tensor is extended by rows and columns containing zeros. Because to this, the size of the data to be processed is increased, which is useful in counteracting the rapid reduction of the size of the interpreted activation map in later convolutional layers. The default area value is zero, which means no rows or columns filled with zeros are added. The Figure on the next page (Fig. 5) depicts how the padding works:

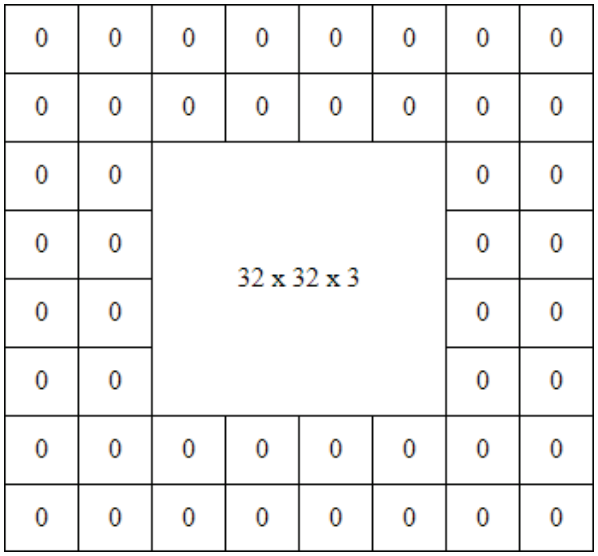


Fig. 5. Graphical representation of the effect of the padding parameter on a matrix subjected to filtration

The Figure above (Fig. 5) shows the 32x32x3 input tensor using an padding of 2. The result of this operation is to increase the input data size to 36x36x3.

You can control the size of the activation map by the number of filters, filter size, stride and padding. The two-dimensional size of the output tensor is described below (2):

$$W_y = \frac{W_e - F + 2O}{K} + 1 \tag{2}$$

where:

W_y – output tensor size
W_e – input tensor size
F – filter size
O – padding
K – stride

4. RECTIFIER LAYER

The rectifier layer adopts the input of the convolutional layer result and introduces the non-linearity of values necessary for proper functioning of the neural network. The following example can be used to understand the importance of nonlinearities in data processing. There is a neural network consisting of two convolutional layers named A and B having one filter and performing operations on the X image. The result of the convolution from the first layer is equal to (3):

$$x'_{i,j} = \sum_{c=1}^{a_l} \sum_{d=1}^{a_w} \left(a_{c,d} x_{(i+c-1),(j+d-1)} \right) \tag{3}$$

However, the result of the second layer of the convolution describes the formula (4):

$$x''_{i,j} = \sum_{c=1}^{b_l} \sum_{d=1}^{b_w} \left(b_{c,d} x'_{(i+c-1),(j+d-1)} \right) \tag{4}$$

What can be written as (5) or (6):

$$x''_{i,j} = \sum_{c'=1}^{b_l} \sum_{d'=1}^{b_w} \left(b_{c',d'} \sum_{c=1}^{a_l} \sum_{d=1}^{a_w} \left(a_{c,d} x_{(i+c-1),(j+d-1)} \right) \right) \tag{5}$$

$$x''_{i,j} = \sum_{c'=1}^{b_l} \sum_{d'=1}^{b_w} \sum_{c=1}^{a_l} \sum_{d=1}^{a_w} \left(x_{(i+c-1),(j+d-1)} a_{c,d} b_{c',d'} \right) \tag{6}$$

This transformation (6) proves that without an additional layer introducing non-linearity of results, the subsequent stages of applying filters can be aggregated into one linear equation. This means that the entire network could be turned into a single layer with a complex filter. In addition, a network that only performs linear operations would not be able to detect nonlinear data dependencies. For this reason, there must be a layer introducing nonlinearity to the output of each convolutional layer.

In convolutional neural networks, the Rectified Linear Unit or ReLU for short, is often used for this task. ReLU functions is described by formula (7):

$$R(x) = \max(0, x) \tag{7}$$

This means that the result of the ReLU function is either zero or an argument of the function depending on the sign of input. It is easy to implement while being one of the best activation functions for convolutional neural networks. Below is a Figure (Fig. 6) comparing the ReLU function graph and two other popular activation functions: sigmoidal and hyperbolic tangent.

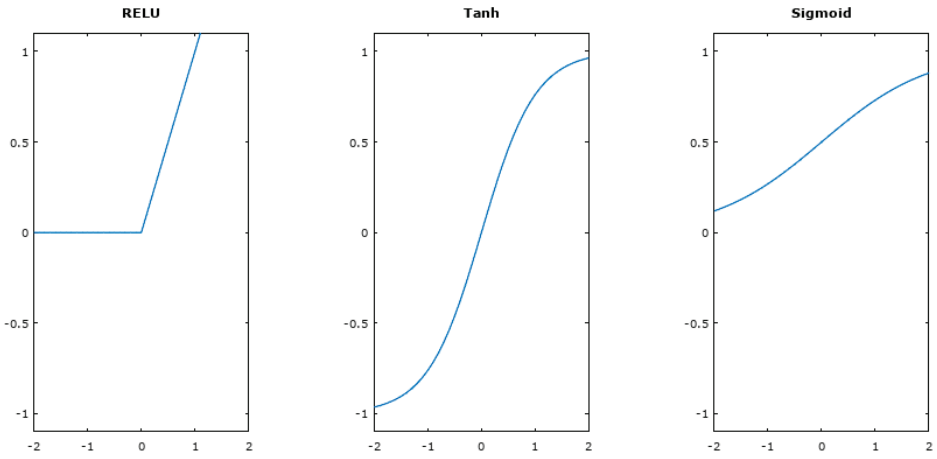


Fig. 6. Graphical comparison of ReLU, hyperbolic tangent function and sigmoid function

The first difference that is easy to see is the simplicity of the ReLU function. Its formula is a comparison of two values, which means ease of implementation and high-performance during computation of the result. For comparison, the formula describing the hyperbolic tangent is (8):

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{8}$$

And the sigmoid function is described by the following formula (9):

$$S(x) = \frac{1}{1 + e^{-x}} \tag{9}$$

Both functions require raising the Euler's number to rational power. This operation is demanding for the processor and carries an approximation error. If the neural network had a lot of convolutional layers, it would also have to have a lot of rectifying layers, which would mean a build up of error.

Another advantage of the ReLU function is the constant derivative. The derivative of the activation function is necessary for correcting the neural network. The first derivative of both sigmoid function and hyperbolic tangent is presented below (Fig.7):

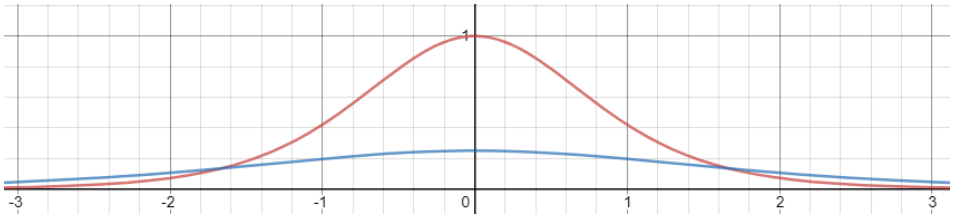


Fig. 7. Graph of the first derivative of sigmoidal function (blue) and hyperbolic tangent (red)

These functions have the first derivatives having very small values on almost the entire range of the X axis. Derivative values are significant only in the vicinity of point 0. This means that if the neuron reaches a value outside this range, the gradient value for network's weight correction would be small and there will be needed a lot of repetitions to change the weights to correct values. This means that the network will have adaptation problems and the learning time will be longer.

The effectiveness of the ReLU function in convolution neural networks intended for image interpretation has been empirically confirmed. Due to its simplicity, the linear unit performs very quickly, and due to the constant value of the first derivative in the $(0;\infty)$ range, the convolutional layers are able to correct their weights in a relatively small number of test iterations.

5. POOLING LAYER

The combination of convolutional layers with rectifying layers is able to effectively recognize images and interpret them. The problem, however, is their dependence on the position of neurons in individual layers. This dependence makes the sum of layers without additional ways of processing the results prone to errors caused by graphic shifts. It is because changing the object's position in the image by just one pixel in any direction changes the entire activation map of all the convolutional layers.

The solution to this problem is adding the pooling layer. It reduces the size of the activation map by aggregating the result value in a given window to one result. An example of a function performed by the pooling layer is to average the values in the window or select the maximum result. The Figure (Fig. 8) describing the operation of the pooling layer choosing the maximum value. The Fig. 8 shows a 4x4 square matrix that has been exposed to the entry of a pooling layer with a 2x2 size window. This means that the input tensor is divided into 4 2x2 areas and the maximum value is taken from each area. In the case of the averaging function, the result would be the average value of the elements of the area.

The pooling layer thus reduces the size of the resulting tensor. This is a desirable effect because interpreted images are huge tensors and the neural network must reduce them as much as possible while maintaining the features that are important from the point of view of the performed action. The pooling layer often has 2x2 selection areas, which means that the size of the input tensor is reduced four times. Such a reduction also has a good effect on reducing the dependence of the neural network on the pixel position of the image, because the smaller the activation map at the entrance of the convolutional layer, the less important the place where the neuron is.

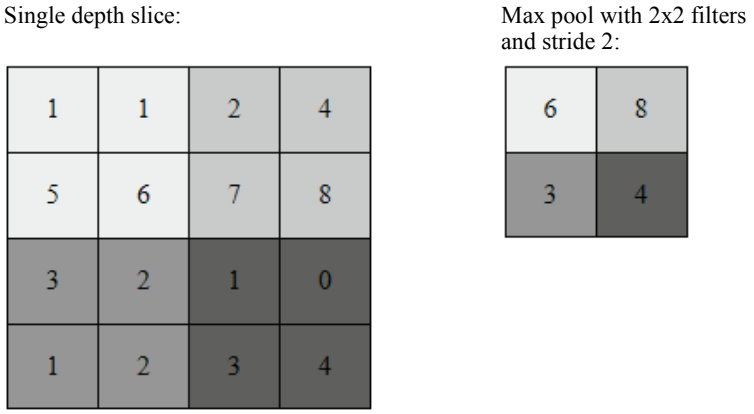


Fig. 8. Presentation of the operation of the pooling layer choosing the largest value in the 2x2 range

A common case of using the pooling layer is to introduce it between several groups of connected convolutional and rectifying layers. The Figure below (Fig. 9) shows an example architecture of a convolutional neural network, in which the pooling layer is first added after two connected layers of convolutional with rectifying layers, and the next pooling layer after one such group.

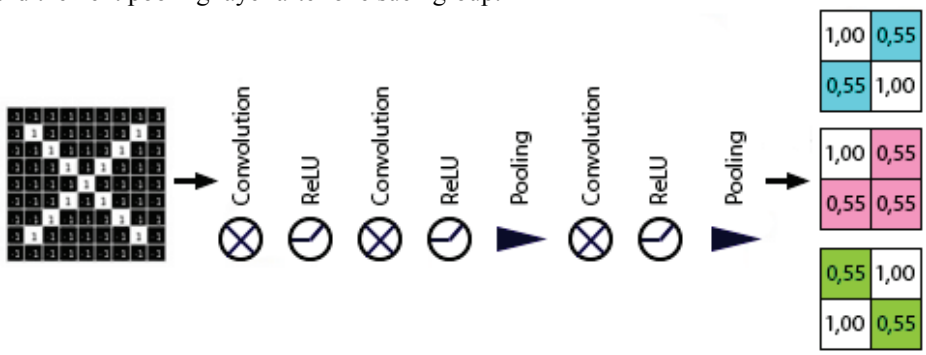


Fig. 9. Example architecture of a convolutional neural network

6. FULLY CONNECTED LAYER

Fully connected layers (FC for short) also called dense layers are located at the end of the convolutional neural network. The three previous types of layers are used to extract significant parts of the image and save them in the least numerous, one-dimensional feature vector. The task of FC layers is to classify the image based on the results of earlier layers.

The fully connected layer is called the dense layer because, unlike the convolutional layers, each of its neurons gets the entire transmitted lower layer result vector. This means that all neurons are fully connected to the entrance of the layer. Often, several FC layers are used connected to each other for better image classification. The last layer of FC in the network determines what class is the object represented in the image. As a rule, this is accomplished by assigning to each class one neuron informing what is the degree of similarity of the analyzed graphic to a particular semantic group. An example of FC layers connected together is shown in the Fig. 10.

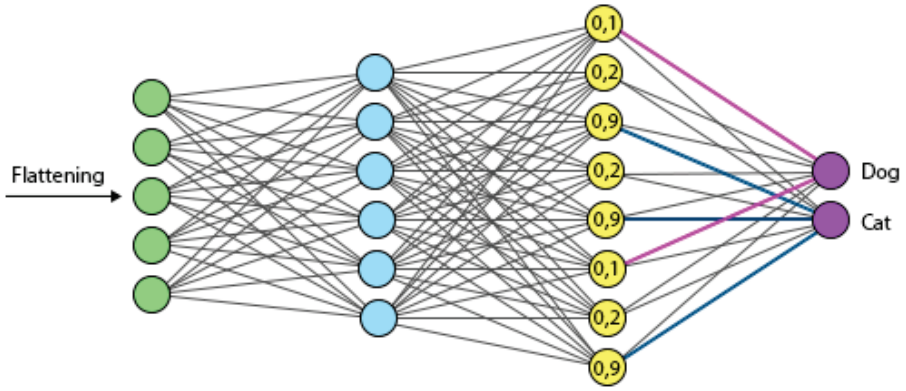


Fig. 10. Example of three layers *Fully Connected*

7. CNN NETWORK ARCHITECTURE

In order to design the convolutional architecture of the neural network, it is necessary to determine, among others, how many layers of particular types should be created, how to connect them, how many neurons should be in each convolutional and fully connected layer, what window sizes will be best in the convolutional and pooling layers and how many filters should contain each.

One of the first proposed Convolutional Neural Network architectures proposed was AlexNet. In 2012, this network won the ILSVRC competition by achieving accuracy in classifying images from a thousand different classes at 84.7%. For comparison, the second place was taken by an algorithm that is not a CNN network with an accuracy of 73.8%. Current network architectures can achieve accuracy above 95%. However, due to its simplicity, the AlexNet network will be discussed. To simplify the description, individual rectifying layers will not be listed, because by default each convolutional layer enters the rectifying layer input. Below is a Table 1 showing a parametric description of each layer in the network.

Table 1. Presentation of subsequent layers of the AlexNet network

AlexNet Netowrk – Structural Details													
Input			Output			Layer	Stride	Pad	Kernel		in	out	Params
227	227	3	55	55	96	Conv1	4	0	11	11	3	96	34944
55	55	96	27	27	96	Maxpool1	2	0	3	3	96	96	0
27	27	96	27	27	256	Conv2	1	2	5	5	96	256	614656
27	27	256	13	13	256	Maxpool2	2	0	3	3	256	256	0
13	13	256	13	13	384	Conv3	1	1	3	3	256	384	885120
13	13	384	13	13	384	Conv4	1	1	3	3	384	384	1327488
13	13	384	13	13	256	Conv5	1	1	3	3	384	256	884992
13	13	256	6	6	256	Maxpool5	2	0	3	3	256	256	0
						Fc6			1	1	9216	4096	37752832
						Fc7			1	1	4096	4096	16781312
						Fc8			1	1	4096	1000	4097000
													Total: 62 378 344

8. LOSS FUNCTION – USE OF SOFTMAX FUNCTION AND CROSS ENTROPY METHOD

Convolutional neural networks with supervised learning need a loss function, which estimates the error in image processing. The result of the loss function is used in the weights correction process. The resulting vector exposed to the output of the last FC layer determines the belonging of the object represented by the processed graphics to one of the defined classes. In the process of teaching a neural network, the loss function checks the numerical result for each group of objects stored in the result vector and compares it with the correct class specified by the tester. Based on the result of the comparison, the partial result of the loss function is calculated for each class so that they are then aggregated into one value of the function loss.

One of the types of loss functions used in neural networks is the Softmax function together with the cross-entropy method. The Softmax function transforms a vector of any element values into a vector of numbers from the $(0; 1)$ range, whose sum of all elements is equal to 1. The Softmax function is used to calculate the normalized partial loss.

The following formula describe Softmax function:

$$f_i(z) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{10}$$

where:

- z – resulting vector
- i – index of the element in the result vector z, for which grated softmax is calculated
- $f_i(z)$ – result of the softmax function for the i-th class from the result vector z
- K – size of the resulting set z

Using this formula, the normalized partial loss can be calculated for each class represented by the result vector. These values should then be aggregated using the cross entropy method. This method calculates the entropy value of information between the expected probability distribution p and the obtained distribution q. There is a formula (11) for this method:

$$H(p, q) = -\sum_{x=1}^{K} p(x) \log\big(q(x)\big) \tag{11}$$

where:

- p – expected probability distribution,
- q – probability distribution calculated using the Softmax function,
- K – size of both vector p and vector q.

An example of the loss function: there is a neural network with a task to determine whether the photo depicts an object belonging to class A or B. The last FC layer of the neural network determined the belonging of the input image to individual classes using the following vector (12):

$$z = [5.45, 3.21] \tag{12}$$

It means that the network recognized the similarity of the object represented by the photo to the class A collection is 5.45, and the similarity to the class B collection is 3.21. After normalizing the scoring using the softmax function, a partial loss vector q was created equation (13) and value (14):

$$q = \left[\frac{e^{5.45}}{e^{5.45} + e^{3.21}}, \frac{e^{3.21}}{e^{5.45} + e^{3.21}}\right] \tag{13}$$

$$q = [0.903784, 0.096216] \tag{14}$$

The object shown in the picture belonged to class A. Because in this case the object can have only once class label, it means that the object does not belong to class B. The expected probability distribution is (15):

$$p = [1.0] \tag{15}$$

The value of cross entropy for vectors p and q is shown in (16):

$$\begin{aligned} H(p,q) &= -(1 * \log 0.903784 + 0 * \log 0.096216) \\ H(p,q) &= 0.043935 \end{aligned} \tag{16}$$

The value of cross entropy is equal to 0.043935. The result of the loss function for the processing of a given image is equal to the value of cross entropy plus a regulating factor. This article describes the method of regulating performing Dropout method, which does not need to add a regularization penalty, so this factor is set to zero. Weight regularization will be described later. The formula for the final value of the loss function is presented in equation (17):

$$\begin{aligned} L &= H(p,q) + \lambda R(W) \\ L &= 0.043935 + 0 \\ L &= 0.043935 \end{aligned} \tag{17}$$

where:

R – function regulating the weights tensor,
W – weights tensor,
λ – constant coefficient determining the importance of weights regularization.

9. WEIGHTS REGULARIZATION – APPLICATION OF THE DROPOUT METHOD

Regularization is a process aimed at preventing over-matching of the neural network to test data. To achieve this goal, the neural network in the learning process should correct the weights in a way that simultaneously reduces the value of the loss function and favors the use of as many image features as possible instead of focusing on a small amount of graphics' areas. One way is to introduce a penalty for having significantly higher numbers in the tensor than others. This task is performed by L2 Normalization with the formula (18):

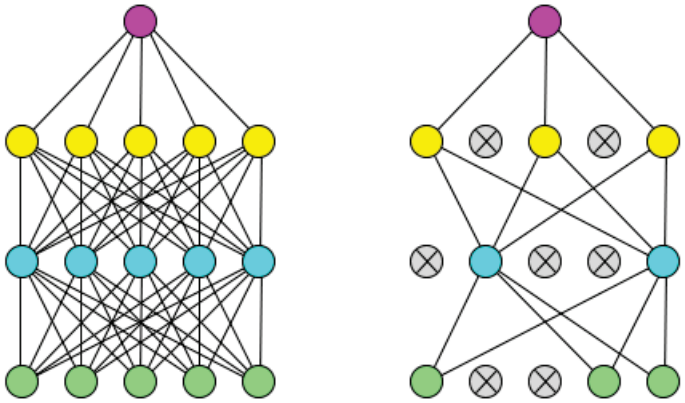
$$R(W) = \sum_{i=1}^{k} \sum_{j=1}^{l} W_{i,j}^2 \tag{18}$$

where:

$R(W)$ – result of regularization penalty
W – weights matrix
k, l – weights matrix sizes

The penalty before adding to the cross entropy result is multiplied by the factor λ. It defines the importance of regularization penalty and how much it forces change on the neural network.

Another way to introduce over-matching protection is by using the Dropout method. It is often used in convolutional neural networks and it consists in the fact that during network training, each neuron has an arbitrarily defined chance of being zeroed for the duration of one test data processing. It means that each neuron can return the number 0 instead of its true value, what is an equivalent of not having effect on image interpretation. The Fig. 11 shows the idea of the Dropout method:



a) Standard Neural Net b) After aplying droput

Fig. 11. Graphical representation of the effects of the Dropout function

The effect of this operation is to force the neural network to involve more neurons to process information. The use of this method extends the time needed for network training, but it increases its effectiveness in interpreting data outside the test range.

10. TRAINING THE CNN NETWORK – APPLICATION THE STOCHASTIC GRADIENT DESCENT

The neural network learning process can be defined as the search for a weights tensor that minimizes the value of the loss function. In order to train the network as quickly as possible, it is necessary to use the optimal search method global minimum. The Stochastic Gradient Descent is effective for this purpose.

This method consists in randomly selecting a small set of test data from the large pool of available samples and performing the Gradient Descent optimization method on them instead of the entire test data set. The Gradient Descent method used in neural networks is to calculate partial derivatives of the loss function relative to the weights matrix and the regularization penalty, and then perform back propagation of error into the lower layers to compute the error gradient on them. The first step is to compute the derivative of the loss function with the formula (19):

$$L = -\sum_{x=1}^{K} p(x)\log\big(q(x)\big) + \lambda R(W) \tag{19}$$

It means that the derivative of the loss function is given by the formula (20):

$$\frac{\partial L}{\partial W_{i,j}} = \frac{\partial\big(-\sum_{x=1}^{K} p(x)\log\big(q(x)\big)\big)}{\partial W_{i,j}} + \frac{\partial\big(\lambda R(W)\big)}{\partial W_{i,j}} \tag{20}$$

Where $W_{i,j}$ is the cell from the *i-th* row and *j-th* column of the weight matrix belonging to the first tier in the network. In this case, the regularization penalty is zero, so the gradient of the loss function can be simplified (21):

$$\frac{\partial L}{\partial W_{i,j}} = -\sum_{x=1}^{K} \frac{\partial(p(x)\log\big(q(x)\big)}{\partial W_{i,j}} \tag{21}$$

The described neural network classifies objects belonging to only one class. This means that the vector p contains one value 1 and the remainder 0. This allows to further reduce the gradient calculations (22):

$$\frac{\partial L}{\partial W_{i,j}} = -\frac{\partial(\log\big(q_i(x)\big)}{\partial W_{i,j}} \tag{22}$$

Where i is the index indicating the class the object should get. Because the derivative of the natural logarithm is equal to (23):

$$\frac{\partial \log(x)}{\partial x} = \frac{1}{x} \tag{23}$$

And derivatives can be combined according to the principle (24):

$$\frac{\partial b}{\partial t} = \frac{\partial b}{\partial a}\frac{\partial a}{\partial t} \tag{24}$$

The derivative of the loss function can be shortened to the form (25):

$$\begin{aligned} \frac{\partial L}{\partial W_{i,j}} &= -\frac{\partial \log\big(q_i(x)\big)}{\partial q_i(x)}\frac{\partial q_i(x)}{\partial W_{i,j}} \\ \frac{\partial L}{\partial W_{i,j}} &= -\frac{1}{q_i(x)}\frac{\partial q_i(x)}{\partial W_{i,j}} \\ \frac{\partial L}{\partial W_{i,j}} &= -\frac{1}{q_i(x)}\frac{\partial q_i(x)}{\partial x_j}\frac{\partial x_j}{\partial W_{i,j}} \end{aligned} \tag{25}$$

The q stands for Softmax function with the formula (26):

$$q_i(x) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}} \tag{26}$$

The derivative of the softmax function for $i = j$ is equal to (27):

$$\frac{\partial q_i(x)}{\partial x_j} = q_i(x)\big(1 - q_j(x)\big) \tag{27}$$

while for $i \neq j$ (28):

$$\frac{\partial q_i(x)}{\partial x_j} = -q_i(x)q_j(x) \tag{28}$$

At this point of consideration, an unknown element is the expression (29):

$$\frac{\partial x_j}{\partial W_{i,j}} \tag{29}$$

The vector x is the output vector of the last FC layer. Each number in it is the result of computation of its neuron represented by the formula (30):

$$x_j = \max\left(0, \sum_{a=1}^{\overline{W'}} x'_a W'_a\right) \tag{30}$$

where:

W' – weight matrix of the last FC layer
x' – the resulting vector of the lower layer, whose results are input for this layer

This means that the gradient (30) for $x_j > 0$ can be written as (31):

$$\frac{\partial x_j}{\partial W_{i,j}} = \sum_{a=1}^{\overline{W'}} \frac{\partial x'_a W'_a}{\partial x'_a} \frac{\partial x'_a}{\partial W_{i,j}} \tag{31}$$

The derivative of the loss function relative to the weight matrix of the first layer of the neural network now depends on the sum of the derivatives of multiplication of individual lower layer outputs by the penultimate layer weights with the weights of the last layer relative to the penultimate layer result vector and the derivative of the penultimate layer outputs relative to the matrix of the last layer weights. The first term can already be calculated, because the values of the vectors W' and $x'$are known at this stage of the computation. This derivative is equal to (32):

$$\nabla L_x = -\frac{1}{q_i(x)} \frac{\partial q_i(x)}{\partial x_j} \sum_{a=1}^{\overline{W'}} \frac{\partial x'_a W'_a}{\partial x'_a}$$
$$\nabla L_x = -\frac{1}{q_i(x)} \frac{\partial q_i(x)}{\partial x_j} \sum W' \tag{32}$$

For $i = j$:

$$\nabla L_x = -\Big(1 - q_j(x)\Big) \sum W' \tag{33}$$

Whereas for $i \neq j$:

$$\nabla L_x = \Big(1 - q_j(x)\Big) \sum W' \tag{34}$$

The value ∇L_x is the so-called local error gradient. The gradient $\frac{\partial x'_a}{\partial w_{i,j}}$ can be calculated by back propagating it to the lower layer. The Gradient method consists in the next calculation of local gradients and the propagation of the remaining error gradient further up to the first layer, in which the process of calculating the gradient of the loss function ends. After calculating all local gradient values, you can make corrections in the weights. The correction is the result of multiplying the local gradient value with the learning step. The learning step is an empirically determined parameter for a given task. Formula for correction (35):

$$k = \nabla L_{x_i} s \tag{35}$$

where:

$s -$ learning step.

11. SUMMARY

Convolutional Neural Networks are able to perform many tasks in the field of image processing with a minimum amount of processing of input data and special algorithms written specifically to extract features of a given class. This means that the CNN network can approach the problem in an abstract way, enabling the use of one network architecture to classify many types of objects. These networks are both flexible and precise. Scientists and engineers are increasingly turning to this type of network to solve problems in the field of image, signal and natural language processing. These networks have only been popular for several years, so one can expect dynamic development of this field.

BIBLIOGRAPHY

[1] A Beginner's Guide To Understanding Convolutional Neural Networks Part 2 https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/
[2] A Gentle Introduction to Pooling Layers for Convolutional Neural Networs https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks
[3] Application of the Cross Entropy Clustering method in biometrics https: //www.mini.pw.edu.pl/~homenda/common/Dr_Krzysztof_Misztal.pdf
[4] Applying Gradient Descent in Convolutional Neural Networks – Nan Cui https://iopscience.iop.org/article/10.1088/1742-6596/1004/1/012027/pdf
[5] Classification and Loss Evaluation – Softmax and Cross Entropy Loss https://deepnotes.io/softmax-crossentropy
[6] CNN Architectures: LeNet, AlexNet, VGG, GoogLeNet, ResNet and more, https://medium.com/@sidereal/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5
[7] CNN – Fully Connected Layer (FC), http://sciagaprogramisty.blogspot.com/2018/03/fully-connected-layer-fc-warstwa-w-peni.html
[8] Convolutional Neural Networks (CNN): Softmax & Cross-Entropy https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-softmax-crossentropy

[9] CS231n Convolutional Neural Networks for Visual Recognition http://cs231n.github.io/linear-classify/#softmax
[10] Derivative of the Sigmoid function, https://towardsdatascience.com/derivative-of-the-sigmoid-function-536880cf918e
[11] Difference between AlexNet, VGGNet, ResNet and Inception https://towardsdatascience.com/the-w3h-of-alexnet-vggnet-resnet-and-inception-7baaaecccc96
[12] Dropout in (Deep) Machine learning https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5
[13] Hope T., Resheff Y.S., Lieder I., 2017. Learning TensorFlow. O'Reily Media.
[14] Lecture Collection, Convolutional Neural Networks for Visual Recognition (Spring 2017) – Fei-Fei Li, Justin Johnson, Serena Yeung – https://www.youtube.com/playlist?list=PL3FW7Lu3i5JvHM8ljYj-zLfQRF3EO8sYv
[15] Notes on Backpropagation – Peter Sadowski https: //www.ics.uci.edu/~pjsadows/notes.pdf
[16] Proofs of Derivatives of Hyperbolics, http://math2.org/math/derivatives/more/hyperbolics.htm

ZASTOSOWANIE KONWOLUCYJNYCH SIECI NEURONOWYCH DO PRZETWARZANIA I INTERPRETACJI OBRAZÓW

Streszczenie

Artykuł ten opisuje zastosowanie Konwolucyjnych Sieci Neuronowych w przetwarzaniu obrazów. W celu lepszego zrozumienia tematu opisano sposób działania sieci. Przedstawiono sieci wielowarstwowe, rodzaje funkcji aktywacji, przykład architektury sieci AlexNet. W artykule skupiono się na opisaniu wykorzystania funkcji straty oraz metody entropii krzyżowej do obliczenia straty w czasie testów. Opisano również sposoby normalizacji wag L2 i Dropout oraz optymalizację funkcji straty za pomocą Stochastycznego Spadku Gradientu.

Słowa kluczowe: Konwolucyjne Sieci Neuronowe, przetwarzanie obrazów, filtracja, splot, funkcja aktywacji, funkcja straty, softmax, entropia krzyżowa, L2, Dropout, Stochastyczny Spadek Gradientu