Ciencia Latina
Revista Multidisciplinar

DOI: https://doi.org/10.37811/cl_rcm.v6i5.3158

Occult mathematics under the learning process in convolutional neural networks

Ricardo Borja-Robalino
ricardostalinborjar@gmail.com
https://orcid.org/0000-0002-3899-1140

Antonio Monleón-Getino
amonleong@ub.edu
https://orcid.org/0000-0001-8214-3205

José Rodellar
jose.rodellar@upc.edu
https://orcid.org/0000-0002-1514-7713

ABSTRACT

In the last decade, artificial intelligence has transformed the world. Big Data and large software companies are prompting researchers to create new algorithms that surpass human intelligence more quickly and efficiently. In 2012 convolutional neural networks (CNN) captured the attention of researchers on the subject of image recognition; becoming popular and efficient. However, the lack of information on a meticulous mathematical process and the tendency of authors to describe this method as a black box, implied that a defined architecture, parameters and hyperparameters generate results with an unknown internal process. The present study develops the detailed mathematical model of forward and backward propagation in a CNN for three-dimensional images (in color), providing researchers with solid tools when developing optimizations in the algorithm. In addition, the models were applied to a proposed architecture that allowed recognizing the stages of the process, the number of network learning parameters and complexity focused on computational expenditure. Finally, tables are included with the most used cost, activation and optimization functions that allow the reader to formulate their own model depending on the architecture, functions and hyperparameters chosen.

Keywords: *convolutional neural networks; mathematical model; forward and backward propagation; cost function; optimization function.*

Correspondencia: ricardostalinborjar@gmail.com
Artículo recibido 10 agosto 2022 Aceptado para publicación: 10 septiembre 2022
Conflictos de Interés: Ninguna que declarar

Cómo citar Borja Robalino, R., Monleón Getino, A., & Rodellar, J. (2022). Matemática oculta bajo el proceso de aprendizaje en redes neuronales convolucionales. *Ciencia Latina Revista Científica Multidisciplinar*, *6*(5), 1031-1063. https://doi.org/10.37811/cl_rcm.v6i5.3158

Matemática oculta bajo el proceso de aprendizaje en redes neuronales convolucionales

RESUMEN

En la última década la inteligencia artificial ha transformado el mundo. El Big Data y grandes empresas de software impulsan a investigadores a crear nuevos algoritmos que superan la inteligencia humana con mayor rapidez y eficiencia. En el año 2012 las redes neuronales convolucionales (CNN) captaron la atención de investigadores en el tema del reconocimiento de imágenes; volviéndose populares y eficientes. Sin embargo, la falta de información de un proceso matemático minucioso y la tendencia de autores a describir este método como una caja negra, implicaron que una arquitectura, parámetros e hiperparámetros definidos generen resultados con un proceso interno desconocido. El presente estudio desarrolló un modelo matemático detallado de forward y backward propagation en una CNN para imágenes tridimensionales (a color), dotando a investigadores de herramientas solidas al momento de generar optimizaciones en el algoritmo. Además, los modelos se aplicaron a una arquitectura planteada que permitió reconocer las etapas del proceso, cantidad de parámetros de aprendizaje de la red y complejidad enfocada al gasto computacional. Se incluye tablas con funciones de costo, activación y optimización más utilizadas que permitan al lector formular su propio modelo dependiendo de la arquitectura, funciones e hiperparámetros seleccionados.

***Palabras clave**: redes neuronales convolucionales; modelo matemático; propagación hacia adelante y atrás; función de costo; función de optimización.*

INTRODUCTION

The rapid technological growth has become the artificial intelligence (AI) and its machine learning algorithms in the best option when developing systems which executes actions of decision, reasoning, problem solving and pattern learning or characteristics in similar pattern to the human brain even with greater precision (Namikawa et al., 2020),(Pesapane et al., 2020),(Ghosh & Kandasamy, 2020).

One of the algorithms currently blooming is the artificial neural networks (ANN) and especially the convolutional neural networks (CNN) as the result of highly efficiency and effectiveness (LeCun et al., 2015). The ANN and CNN usage became popular in the tech giants such as Google, Facebook, Amazon, Microsoft and others. Since 2009, Google has used ANN in the captchas problems through the reCAPTCHA application. Stanford University managed to generate captions automatically meanwhile Microsoft developed BrainMaker application to attract future buyer customers (Van Houdt et al., 2020). Additionally, there are applications addressed to translations, personal assistants, freelance vehicles among others (Perconti & Plebe, 2020).

The existence of various classification algorithms such as: logistic regression, support vector machine, decision trees, XGBoost, random forests, neural networks and deep learning, offers numerous alternatives to researches depending on the type of problem to be solved (Lazzeri, 2020). However, CNN's (deep learning) epic breakthrough in the computer area by vision was in 2012 where a contestants group led by Geoffrey Hinton won first place in the “ImageNet” image classification contest using the AlexNet architecture and achieving results that once again captured the attention of researchers in this algorithm (Krizhevsky et al., 2017).

Various authors describe the CNN as algorithms whose process based on logic and mathematical models are presented as a kind of black box where their responses must be accepted regardless of unfamiliarity of their process (Buhrmester et al., 2019)(Sturm et al., 2016)(Baselli et al., 2020). From the mathematical and scientific perspective, it is important the researcher possesses the necessary tools and information in relation to this type of network, in such a way that it allows them to analyze the architecture and its established learning process; recognizing critical points of failure in the algorithm and allowing the possibility of optimizing the posed methodology and programming.

The present research aims to demystify CNNs as black boxes, through the detailed mathematical development of learning process of a network (forward and backward propagation), providing researchers with a solid and complete mathematical knowledge of this type of networks developed for the three-dimensional case, considering that there is no detailed information only for the two-dimensional case. In addition, the approach of the models on an architecture of a CNN is exemplified; applying a standard activation, cost and optimization function. Finally, tables are presented with the most used functions, easing readers the possibility of introducing any of them to the model depending on the need.

THEORETICAL BACKGROUND

For the mathematical development, the basic knowledge about: convolutional neural network, mathematical formulas for the convolution process, introduction of non-linearity, loss functions, optimization and network learning process were considered.

Convolutional Neural Networks (CNN)

Convolutional neural networks (CNN) seek to imitate the behavior of the human visual cortex, which processes simple and highly complex images at extraordinary speed (Asghar et al., 2019). CNNs are quite similar to artificial neural networks with the following differences: the network assumes an image directly as input, a convolution operation is executed between the input matrix and the established kernel, there is a pooling layer that allows reducing the dimension and thus the number of parameters (Wadawadagi & Pagi, 2020).

CNNs are generally composed of three essential layers allowing the network to learn different levels of abstraction. The first is a convolution layer, in many cases followed by a pooling layer and finally a fully connected layer (exactly the same as an ANN).

Layers

Convolutional Layer

The convolution operation (Equation 1) allows the network to learn spatial hierarchies of local patterns, working under regions of established dimensions by the user, which facilitates the detection of certain visual features such as contours, edges, color drops, lines, etc.(Dhillon & Verma, 2020). This process allows recognizing a certain characteristic in a specific place in the image and then being able to detect it in any location it is being found, unlike ANNs that only learn global patterns. In this layer the parameters are a set

of three-dimensional filters that are adjusted as the network learns, allowing data to be transformed in such a way that certain characteristics become more dominant in the output image (Minaee et al., 2020). Its mathematical formulation is presented in the following equation:

$$C = g\,[W\ * X + b]$$

where: *W* = weight or kernel*; X* = input matrix*; b* = bias*;* $*$ = convolution operation*; g* = activation function*.*

Pooling layer

They are often located after a convolutional layer and their function is to generate a simplification process or parameters reduction progressively, avoiding over-fitting the network. As a result, a region with condensed information is obtained, maintaining the input spatial relationship (Lopez-Pinaya et al., 2020). This type of layer generally works under two possible options:

- Max-pooling (Equation 2). - It is the most widely used, the size of the sub- region is variable for convenience, its objective is to find the maximum value a two-dimensional window sample and pass it as a result. Its process is very similar to that used by the human visual cortex (Wu, 2017). (2)

  $P_{max} = \max(C_{i,j})$; $C_{i,j}$= *two-dimensional window in C output.*

- Average-pooling (Equation 3). – unlike max-pooling, its result comes from the mean (average) value within the two-dimensional window (Lee et al., 2009; Wu, 2017).

$$P_{average} = \frac{1}{Po}\sum(C_{i,j})$$

where: *Po = pooling window size;*

Fully or densely connected layer

In this layer, the operating process is similar to an artificial neural network (Equation 4), using the perceptron as the fundamental unit. The sum of inputs multiplied by the associated weights and added a bias (polarization) constitutes the neuron linear activation (Martineau et al., 2020). The non-linearity to the system is introduced through an activation function, becoming in this case (CNN) in the final phase of image classification. A convolutional network architecture can contain one or more fully connected layers. The convolutional layer as well as the densely connected one have an identical functional form which allows conversions between them, taking into account

that in the first case the connections are made to a local region in the input while in the second one works with the global region (Dhillon & Verma, 2020)(Kim, 2014).

$\hat{Y} = g[W \cdot X + b]$; $\cdot$ = matrix product

Parameters and hyperparameters

In a CNN, the parameters constitute those variables internal to the mathematical model of the network (weights and biases), which can be estimated or learned from the input data. Hyper-parameters are external elements configuration adjusting the learning process and are established by the programmer through experience and intuition depending on the type of architecture or problem to be analyzed. Among the best known hyper-parameters at the structure, topology or algorithm level we have: number of layers, function activation functions, loss functions, neurons number, learning rate, decay, momentum, nesterov, epochs, batch size, dilation, dropout, normalization, kernel size, filter numbers, padding, stride, etc (Rawat & Wang, 2017).

Funciones de costo

It allows to quantify the error (cost) coming between the expected and predicted output during the network learning process, in short; assesses how well the algorithm predicts (Janocha & Czarnecki, 2017). Loss functions can be regression or classification; among the most used are (See Table 1):

Table 1. *Cost Functions.*

Identification	Formula	Characteristics
Regression		
Mean squared errors (MSE)(Parmar, 2018)	$E = \frac{\sum_{I=1}^{n}(\widehat{Y}_i - Y_i)^2}{2n}$ (5) $\widehat{Y}_i = predicted \;\; ; \;\; Y_i = real$	It is the average of the squared difference between the predicted and actual values.
Mean absolute errors (MAE)(Parmar, 2018)	$E = \frac{\sum_{i=1}^{n}\lvert\widehat{Y}_i - Y_i\rvert}{n}$ (6) $\widehat{Y}_i = predicted \;\; ; \;\; Y_i = real$	It is the average of the absolute differences between the predicted and actual values.
Binary classification		
Binary cross-entropy (Log loss) (Parmar, 2018).	$E = -\sum_{i=1}^{n}[Y_i \cdot \log(\widehat{Y}_i) + (1 - Y_i) \cdot \log(1$ (7) $Y_i = real \;\; ; \widehat{Y}_i = predicted$	It is commonly used where the loss increases as the predicted probability differs from the real one, also called negative logarithmic probability.
Multiclass classification		
Multi-class cross entropy (categorical-crossentropy) (Haykin, 2009).	$E = -\sum_{i=1}^{n}\sum_{j=1}^{c} Y_{ij} \cdot \log(\widehat{Y_{ij}})$ (8) $\widehat{Y_{ij}}$ = Probability of i_{th} element in j.	It is a generalization of binary cross entropy. Y_{ij} = 1 If the i_{th} element is in j Y_{ij}= 0 other case.
Kullback-leibler divergence ($D_{KL}(P\|\|Q)$) (Kullback & Leibler, 1951).	$-\sum_{i=1}^{n}\sum_{j=1}^{c}[P(Y_{ij}) \cdot \log Q(Y_{ij}) - P(Y_{ij}) \cdot \log P(Y_{ij})]$ (9) $P(Y_{ij})$ = Real probability of i_{th} element in j. $Q(Y_{ij})$ = Predicted probability of i_{th} element in j.	Similar to multi-class entropy and is called relative entropy of P with respect to Q. If the divergence is 0, distributions are equal.

Activation functions (FA)

The activation functions are used in each convolutional and densely connected layer, introducing non-linearity to the mathematical model (activation potential z) of the posed architecture [26]. The process is necessary knowing that an image has many non-linear elements. This type of model is not subject to the superposition principle, where a linear problem might be decomposed into 2 or more simpler sub-problems (Berzal, 2018). FAs are chosen depending on the task that the neurons must perform; Among the widely used are (See Table 2):

Table 2. *Activation functions*.

Identification	Formula	Derivative	Characteristics
Sigmoid(Berzal , 2018).	$g(z) = \frac{1}{1+e^{-}}$ (10)	$g'(z) = g(z)[1-g($ (11)	g(z) tends to 0 when z tends to $-\infty$ and tends to 1 when tends to $+\infty$. The function range is: $(0,1)$.
Hyperbolic tangent (TanH)(Berzal, 2018).	$g(z) = \frac{e^{z}-e^{-z}}{e^{z}}$ (12)	$g'(z) = 1 - g(z)^2$ (13)	It is a Sigmoid's re-scaled version and has higher derivatives, which contributes to learning. The function range is: $[-1,1]$.
Rectified linear unit (ReLU)(Gu et al., 2018).	$g(z) = \max(0,z)$ (14) $g(z) = \begin{cases} 0; & for\ z < 0 \\ z; & for\ z \geq 0 \end{cases}$	$g'(z) = \begin{cases} 0, for\ z < 0 \\ 1, for\ z \geq 0 \end{cases}$ (15)	It is the most used in Deep Learning. The function range is:$[0,\infty)$.
LeakyReLU (Gu et al., 2018).	$g(z) = \max(\epsilon z, z)$ (16) $\epsilon \ll 1$ $g(z)\begin{cases} 0.01z; & for\ z < \\ z & ; for\ z \geq 0 \end{cases}$	$g'(z)\begin{cases} 0.01, for\ z < 0 \\ 1 \quad , for\ z \geq 0 \end{cases}$ (17)	It is similar to ReLU, with the difference that for negative values there is a constant slope (generally 0.01). The function range is:$[0,\infty)$.
SoftPlus(Glorot et al., 2011).	$g(z) = \ln(1+e^{z})$ (18)	$g'(z) = \frac{1}{1+e^{-z}}$ (19)	It is a ReLU's smooth approximation and is distinguishable throughout. In some cases, it is used in the last layer. The function range is: $(-\infty,+\infty)$.
Softmax(Zhang et al., 2018).	$g_i(z) = \frac{e^{z_i}}{\sum_{j=1}^{J} e^{z_j}}$ (20) J = number of classes. $\forall i \in 1 \ldots . J$	$g'(z) = g(z)[1-g(z)]$ $if\ \ i = j$ (21) $g'(z) = g_i(z)\left[1-g_j(z)\right]$ $if\ i \neq j$	It takes as input a vector z and normalizes it to a probability distribution. The function range is: $(0,1)$.

Optimization functions (FO)

The network optimization goal is to minimize the cost function, reducing the error between the desired and predicted output; by estimating and adjusting the model parameters (architecture) (Rao et al., 2018). The FO uses the partial derivatives (gradient) of the loss function evaluated with respect to the current weight $E(W)$, getting iteratively and gradually to determine the best combinations of weights W^* and biases b^* that allows reaching the global minimum of the function where $E(W^*) \preccurlyeq E(W)$ and $E(b^*) \preccurlyeq$

$E(b)$; fulfilling the optimality condition $[\nabla E\ (W^*) = 0]$ where $\nabla = \left[\frac{\partial}{\partial w_1}, \frac{\partial}{\partial w_2}, .. \frac{\partial}{\partial w_n}\right]^T$.

The main algorithm used for optimization in convolutional neural networks is the Stochastic Descending Gradient (Bottou, 2012) (See Table 3).

Table 3. *Stochastic gradient.*

Identification	Formula	Characteristics
Stochastic gradient descent (SGD)(Bottou, 2012)	$W_{t+1} = W_t - \eta\, \nabla(W_t)$ (22) $\eta = learning\ rate$; $\nabla = gradient$ $\nabla(W_t) = \frac{\partial E}{\partial W_t}$; $t = iteration\ (epoch)$	Similar to the descending gradient, with the difference it is updated by small random samples of the training set (mini batches), which allows it to be more scalable and faster.

There are different optimization algorithms for the stochastic descending gradient, among the best known are the following (See Table 4):

Table 4. *Optimization function of SGD.*

Identification	Formula	Characteristics
Momentum (M)(Ruder, 2017)	$M_t = \gamma W_{t-1} - \eta\, \nabla(W_t)$ (23) $\therefore\ W_{t+1} = W_t - M_t$ $\gamma = impulse\ ;\ \gamma < 1$	Speed SGD in the convergence direction and dampen oscillations Generally $\gamma = 0.9$
Nesterov accelerated gradient (NAG)(Ruder, 2017)	$M_t = \gamma W_{t-1} - \eta\, \nabla(W_t - \gamma M_{t-1})$ (24) $\therefore\ W_{t+1} = W_t - M_t$	Optimizes SGD with moments, avoiding very long jumps through a correction in the previous gradient γM_{t-1}. Avoid going too fast and reaching a divergence.
Gradient based algorithm (Adagrad)(Ruder, 2017)	$W_{t+1} = W_t - \frac{\eta}{\sqrt{\varepsilon I + diag(G_t)}} \cdot \nabla_t$ (25) $\nabla_t = \frac{1}{n}\sum_{i=1}^{n} \nabla(W_t)$; $G_t = \sum_{\tau=1}^{t} \nabla_t \nabla_t^T$ $\eta = initial\ learning\ rate$; $\varepsilon \ll 1$ $\nabla_t = gradient\ estimation\ in\ the\ interv$ $G_t = gradient\ estimation\ in\ the\ interv$	Adapt the learning rate through small updates in parameters that act within common and unusual image characteristics. Scales with respect to the accumulated square gradient in each iteration t in each dimension. $\epsilon \approx 1^{-8}$
Adadelta(Ruder, 2017)	$W_{t+1} = W_t - \frac{\eta}{\sqrt{\varepsilon\, I + E[\nabla^2]_t}} \cdot \nabla_t$ (26) $E[\nabla^2]_t = \gamma E[\nabla^2]_{t-1} + (1-\gamma)\nabla_t^2$ $\gamma = Impulse$	It is an Adagrad extension that reduces the monotonically and decreasing aggressive learning rate. Adadelta restricts the accumulated past gradients to a fixed size of the last $E[\nabla^2]_{t-1}$ gradients.

Adaptive learning algorithm (RMSprop) (Ruder, 2017)	$W_{t+1} = W_t - \frac{\eta}{\sqrt{\varepsilon I + E[\nabla^2]_t}} \cdot \nabla_t$ (27) $E[\nabla^2]_t = \rho E[\nabla^2]_{t-1} + (1-\rho)\nabla_t^2$	It is similar to Adadelta. $\rho = forgetting\ factor$ Generally, it recommends $\rho = 0.9$
Adam(Ruder, 2017)	$W_{t+1} = W_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} \hat{m}_t$ (28) $\hat{m}_t = \frac{m_t}{1-\beta_1^t}$; $\hat{v}_t = \frac{v_t}{1-\beta_2^t}$ $m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla_t$ $v_t = \beta_2 v_{t-1} + (1-\beta_2)\nabla_t^2$	Considering as one of the fastest by fastair, combines RMSprop and moments. The authors posed $\beta_1 = 0.9$; $\beta_2 = 0.999\ and\ \epsilon \approx 1^{-8}$; $m_t\ and\ v_t$ are initialized to zero. $m_t = first\ moment\ estimation$ $v_t = second\ moment\ estimation$

How learning is done on CNN?

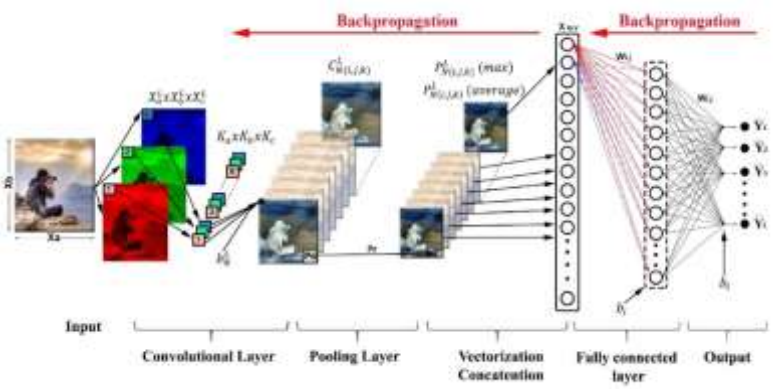
At the beginning of the training, the previously converted and normalized images go through the convolution, pooling and classification (forward propagation) layers, obtaining the prediction of the target of each image or group of images. This is the starting point of learning using the backward propagation process and the cost function propagates the error throughout the entire network (from back to front); by optimizing function. This process allows adjusting and updating network weights and biases in each batch or mini batch established by the user. The parameters adjustment o will allow the network to learn the image characteristics, improving its prediction capacity at each epoch (Liu et al., 2015), (Koushik, 2016).

MODELING AND APPLICATION

Mathematical modeling of the learning process in CNN

The forward y backward propagation process updates the patterns and entries in each convolutional neuronal network layer (See Figure 1).

Figure 1. *Convolutional neural network with mathematical model nomenclature.*



Represents the nomenclature used for modeling in relation to the input, number of channels and number of filters used for the convolution and pooling layers. Similarly for the conversion to be applied in the densely connected layer.

The modeling used the following nomenclature:

- L= number of layers $0 > L \leq M$ (generally made up of a convolution and pooling layer).
- $X_a \cdot X_b \cdot X_c$ = three-dimensional input (large, width and depth), where $X \in \mathbb{R}^{X_a^L \cdot X_b^L \cdot X_c^L}$.
- i, j, k = i, j belongs to the positioning of X in the two-dimensional plane and k is its depth.
- $K_a \cdot K_b \cdot K_c$ = kernel (filters) positioning in its three dimensions.
- s = number of output filters.
- P_o= pooling window positioning (P_x, P_y).
- P_r = pooling reduction scale.
- u , v = vector dimensions in the densely connected layer (rows, columns).
- η = learning rate.
- g = activation function (non-linear operation).
- FC = fully connected layer.
- $\hat{Y}_i$ = number of labels (output classes).
- N = number of output feature maps.

Forward propagation

Forward propagation in a CNN begins with the image, considered as an input matrix X of dimensions $X_a \cdot X_b \cdot X_c$. The extraction of general characteristics to specific was carried out through the different layers established in an architecture, taking into account that:

- The convolution operation was restructured for a color image.

$$C^L_{N\,(i,j,k)} = g\left[\sum_{p=0}^{Ka-1}\sum_{q=0}^{Kb-1}\sum_{r=0}^{K_c-1} W^L_{N\,(p,q,r)} X^L_{N\,(i-p,j-q,k)} + b^L_N\right] ; L = 1,2..M \;;\; N = 1,2 \dots s$$

The non-linearized activation function in the generally is represented like $g[\cdot]$. For the application, it was used the activation functions from Table 2.

- Using and modifying Equations 2 y 3, the pooling layer is represented by:

$$P^L_{N\,(i,j)}(\max) = max\left[\sum_{P_x=0}^{Pr-1}\sum_{P_y=0}^{Pr-1} C^L_{N[\Pr\cdot i-Px,\Pr\cdot j-Py,k]}\right]$$

$$i = 1,2 \dots . \frac{X^L_a}{Pr} \;;\; j = 1,2 \dots . \frac{X^L_b}{Pr} \;;\; k = 1,2 \dots X_c;\; N = 1,2 \dots s$$

$$P^L_{N\,(i,j)}(average) = \frac{1}{P_r \cdot P_r}\left[\sum_{P_x=0}^{Pr-1}\sum_{P_y=0}^{Pr-1} C^L_{N[\Pr\cdot i-Px,\Pr\cdot j-Py,k]}\right]$$

$$i = 1,2 \dots . \frac{X^L_a}{Pr} \;;\; j = 1,2 \dots . \frac{X^L_b}{Pr} \;;\; k = 1,2 \dots X_c;\; N = 1,2 \dots s$$

- Basing on Equation 4, the densely connected layer model is:

$$\hat{Y}_i = g\left[X \cdot {W_{i,j}}^T + b\right]$$

Backward propagation

The backpropagation objective is to update the weights and biases in each layer, starting from back to front with the propagation of the error between the output of the forward propagation $\hat{Y}_i$ and the desired output Y_i. For the mathematical modeling in the present study it was considered:

- Error function: $E = [\cdot]$
- In several convolutional layers the input $X^L_{N\,(i,j,k)}$ of each layer is (Koushik, 2016):

$$X^{L}_{N\,(i,j,k)} = \sum_{p}\sum_{q}\sum_{r} W^{L}_{N\,(p,q,r)} X^{L-1}_{N\,(i-p,j-q,k)} + b^{L}_{N}$$

$$where:\ L = 1,2..M;\ each\ layer\ X^{L-1}_{(i,j,k)} = C^{L-1}_{q\,(i,j,k)}$$

Next, the mathematical models for the convolution, pooling and densely connected layers' backpropagation are developed; considering like optimization function the stochastic descending gradient.

Output layer – densely connected

Weight updating $\boldsymbol{W_{i,j}}$ *in densely connected layer.*

$$W_{actual} = W_{initial} - \eta\nabla(W)$$

Whereby.

$$\nabla W_{i,j} = \frac{\partial E}{\partial W_{i,j}}$$

Applying the chain rule.

$$\nabla W_{i,j} = \frac{\partial E}{\partial \hat{Y}_i}\frac{\partial \hat{Y}_i}{\partial W_{i,j}}$$

$$\nabla W_{i,j} = \sum_{i=1}^{u}\sum_{j=1}^{v}\frac{\partial E}{\partial \hat{Y}_i}\frac{\partial}{\partial W_{i,j}} g\left(\sum_{i=1}^{u}\sum_{j=1}^{v} W_{i,j}X_j + b_i\right)$$

$$\nabla W_{i,j} = \sum_{i=1}^{u}\sum_{j=1}^{v}\frac{\partial E}{\partial \hat{Y}_i}\, d(g) X_j$$

$$\nabla W_{i,j} = \sum_{i=1}^{u}\sum_{j=1}^{v}\Delta e \cdot X_j^T$$

Where:

$$\Delta e = \frac{\partial E}{\partial \hat{Y}_i}\, d(g)$$

The actual weight spread W_{actual} is:

$$W_{actual} = W_{initial} - \eta\nabla(W)$$

Therefore, the weight updating in a densely connected layer would be:

$$W_{actual} = W_{initial} - \eta\left[\frac{\partial E}{\partial \hat{Y}_i}\, d(g) \cdot X_j^T\right]$$

1. *Bias updating $\boldsymbol{b_j}$ densely connected layer.*

$$b_{actual} = b_{initial} - \eta\nabla(b_i)$$

Whereby.

$$\nabla b_i = \frac{\partial E}{\partial b_i}$$

Applying the chain rule, it has:

$$\nabla b_i = \frac{\partial E}{\partial \hat{Y}_i}\frac{\partial \hat{Y}_i}{\partial b_i}$$

$$\nabla b_i = \sum_{i=1}^{u}\sum_{j=1}^{v}\frac{\partial E}{\partial \hat{Y}_i}\frac{\partial}{\partial b_i} g\left(\sum_{i=1}^{u}\sum_{j=1}^{v} W_{i,j}X_j + b_i\right)$$

$$\nabla b_i = \sum_{i=1}^{u}\sum_{j=1}^{v}\frac{\partial E}{\partial \hat{Y}_i}\, d(g)$$

$$\nabla b_i = \Delta e$$

The actual bias spread b_{actual} is:

$$b_{actual} = b_{inicial} - \eta\nabla(b)$$

Therefore, the bias updating in a densely connected layer would be:

$$b_{actual} = b_{initial} - \eta\Delta e$$

Input updating $\boldsymbol{X_{FC}}$ in the densely connected layer.

The input updating X_{FC}^{L} would be:

$$X_{FC.actual}^{L} = X_{FC.initial}^{L} - \eta\nabla(X_{FC}^{L})$$

Whereby.

$$\nabla X_{FC}^{L} = \frac{\partial E}{\partial X_{FC}^{L}}$$

Applying the chain rule, it has:

$$\nabla X_{FC}^{L} = \sum_{i=1}^{u}\sum_{j=1}^{v}\frac{\partial E}{\partial \hat{Y}_i}\frac{\partial \hat{Y}_i}{\partial X_{FC}^{L}}$$

$$\nabla X_{FC}^{L} = \sum_{i=1}^{u}\sum_{j=1}^{v}\frac{\partial E}{\partial \hat{Y}_i}\frac{\partial}{\partial X_{FC}^{L}} g\left(\sum_{i=1}^{u}\sum_{j=1}^{v} W_{i,j}X_j + b_i\right)$$

$$\nabla X_{FC}^{L} = \sum_{i=1}^{u}\sum_{j=1}^{v}\frac{\partial E}{\partial \hat{Y}_i}\, d(g)W_{i,j}$$

$$\nabla X_{FC}^{L} = \sum_{i=1}^{u}\sum_{j=1}^{v}\Delta e \cdot W_{i,j}$$

Therefore, to make the matrix product, the following must be taken into account:

$$\nabla X_{FC}^{L} = \Delta e \cdot W^{T}$$

The input spread X_{FC}^{L} is:

$$X_{FC.actual}^{L} = X_{FC.initial}^{L} - \eta\nabla(X_{FC}^{L})$$

The input updating X_{FC} in a densely connected layer would be:

$$X_{FC.actual}^{L} = X_{FC.initial}^{L} - \eta[\Delta e \cdot W^{T}]$$

Hidden or intermediate densely connected layers

Weigh updating $\boldsymbol{W_{i,j}}$ in densely connected layer.

$$W_{actual} = W_{inicial} - \eta\nabla(W)$$

Whereby.

$$\nabla W_{i,j} = \frac{\partial E}{\partial W_{i,j}}$$

Applying the chain rule.

$$\nabla W_{i,j} = \sum_{i=1}^{u}\sum_{j=1}^{v}\frac{\partial E}{\partial \hat{Y}_i}\frac{\partial \hat{Y}_i}{\partial W_{i,j}} \quad ; \quad where: \hat{Y}_i = X_{FC}^{L}$$

$$\nabla W_{i,j} = \sum_{i=1}^{u}\sum_{j=1}^{v}\nabla X_{FC}^{L}\frac{\partial}{\partial W_{i,j}}g\left(\sum_{i=1}^{u}\sum_{j=1}^{v}W_{i,j}X_{i,j}^{L-1} + b_i\right)$$

$$\nabla W_{i,j} = \sum_{i=1}^{u}\sum_{j=1}^{v}\nabla X_{FC}^{L}\, d(g)X_{i,j}^{L} \quad ; \quad where: \nabla X_{FC}^{L} = \Delta e \cdot \left(W_{i,j}^{L}\right)^{T}$$

$$\nabla W_{i,j} = \Delta e \cdot \left(W_{i,j}^{L}\right)^{T} d(g)X_{i,j}^{L-1}$$

The actual weigh spread W_{actual} is:

$$W_{actual} = W_{initial} - \eta\nabla(W)$$

Therefore, the weigh updating in a densely connected layer would be:

$$W_{actual} = W_{initial} - \eta \left[\Delta e \cdot \left(W_{i,j}^{L}\right)^{T} d(g) X_{i,j}^{L-1}\right]$$

1. *Bias updating $\boldsymbol{b_j}$ densely connected layer.*

$$b_{actual} = b_{initial} - \eta \nabla(b)$$

Whereby.

$$\nabla b_i = \frac{\partial E}{\partial b_i}$$

Applying the chain rule, it is:

$$\nabla b_i = \sum_{i=1}^{u} \sum_{j=1}^{v} \frac{\partial E}{\partial \hat{Y}_i} \frac{\partial \hat{Y}_i}{\partial b_i}$$

$$\nabla b_i = \sum_{i=1}^{u} \sum_{j=1}^{v} \nabla X_{FC}^{L} \frac{\partial}{\partial b_i} g\left(\sum_{i=1}^{u} \sum_{j=1}^{v} W_{i,j} X_j^L + b_i\right)$$

$$\nabla b_i = \sum_{i=1}^{u} \sum_{j=1}^{v} \Delta e \cdot \left(W_{i,j}^{L}\right)^{T} d(g)$$

The actual bias spread b_{actual} is:

$$b_{actual} = b_{initial} - \eta \nabla(b)$$

Therefore, bias updating in a densely connected layer would be:

$$b_{actual} = b_{initial} - \eta \left[\Delta e \cdot \left(W_{i,j}^{L}\right)^{T} d(g)\right]$$

Input updating $\boldsymbol{X_{FC}^{L}}$ in the densely connected layer.

In the backpropagation process, the last convolution or pooling layer becomes the input of the densely connected layer; therefore, updating this input X_{FC}^{L} would be:

$$X_{FC.actual}^{L} = X_{FC.initial}^{L} - \eta \nabla(X_{FC}^{L})$$

$$\nabla X_{FC}^{L} = \frac{\partial E}{\partial X_{FC}^{L}}$$

Applying the chain rule, it is:

$$\nabla X_{FC}^{L} = \sum_{i=1}^{u}\sum_{j=1}^{v} \frac{\partial E}{\partial \hat{Y}_i}\frac{\partial \hat{Y}_i}{\partial X_{FC}^{L}}$$

$$\nabla X_{FC}^{L} = \sum_{i=1}^{u}\sum_{j=1}^{v} \nabla X_{FC}^{L}\frac{\partial}{\partial X_{FC}^{L}} g\left(\sum_{i=1}^{u}\sum_{j=1}^{v} W_{i,j}X_{j}^{L-1} + b_i\right)$$

$$\nabla X_{FC}^{L} = \sum_{i=1}^{u}\sum_{j=1}^{v} \nabla X_{FC}^{L}\, d(g)\left(W_{i,j}\right)^{T}$$

The spread to the input ∇X_{FC}^{L} of a densely connected hidden or intermediate layer is:

$$X_{FC.actual}^{L} = X_{FC.initial}^{L} - \eta\nabla(X_{FC}^{L})$$

Therefore, the input updating X_{FC} e in a densely connected layer would be:

$$X_{FC.actual}^{L} = X_{FC.initial}^{L} - \eta\left[\Delta X_{FC}^{L}\, d(g)\left(W_{i,j}\right)^{T}\right] \quad (39)$$

Vector $\boldsymbol{X_{FC}}$ to matrix $\boldsymbol{X_{N(i,j)}^{L}}$

When considering a pooling layer localization before a densely connected one; the values of X_{FC}^{L} are the result of vectorization and concatenation in the forward propagation process. Executing the inverse process, it obtains a matrix $\mathrm{X}_{\mathrm{N(i,j)}}^{\mathrm{L}}$ that represents pooling layer output.

$$X_{N(i,j)}^{L} = \sum_{N=1}^{X_c}\left(\sum_{i=0}^{X_a-1}\sum_{j=0}^{X_b-1} X_{FC\,(N\cdot X_b+i)}\right) \; ; L = 1,2,\dots M \; ; \; N = 1,2\dots s \; ; \; X_c = 1,2\dots s \quad (40)$$

Pooling layer

- For average pooling case, each average $X_{N(i,j)}^{L}$ returns to all matrix size cells $P_o\ (P_x, P_y)$:

$$P_{N(i,j,k)}^{L} = \sum_{k=1}^{X_c}\left(\sum_{i=0}^{Px-1}\sum_{j=0}^{P_y-1} X_{FC(i,j)}^{L}\right) \; ; L = 1,2,\dots M \; ; \; N = 1,2\dots s \quad (41)$$

- In the case, it is a max-pooling, the maximum will be placed in the position that was extracted in the forward propagation, while the other cells will take a value of zero:

$$P^L_{N(i,j,k)} = \sum_{k=1}^{Xc}\left(\sum_{i=0}^{Xa-1}\sum_{j=0}^{Xb-1} X^L_{FC(i,j)}\right) = X^L_{N(i,j,k)} \quad (42)$$

where: $X^L_{N(i,j,k)}$= extraction position in forward; $X^L_{FC(i,j)}$= maximum value to spread.

By propagating the pooling layer, $P^L_{N(i,j,k)}$ becomes the convolutional layer output $(X^L_{N(i,j,k)} = \nabla X^L_{N(i,j,k)})$

Convolutional layer

1. *Weight updating (Kernel).*

$$W^L_{N(p,q,r)actual} = W^L_{N(p,q,r)initial} - \eta\, \nabla W^L_{N(p,q,r)}$$

Whereby:

$$\nabla W^L_{N(p,q,r)} = \frac{\partial E}{\partial W^L_{N(p,q,r)}}$$

$$\nabla W^L_{N(p,q,r)} = \frac{\partial E}{\partial X^L_{N\,(i,j)}}\frac{\partial X^L_{N\,(i,j)}}{\partial W^L_{N(p,q,r)}}$$

Using a triple summation so that the weight slides over the entire characteristics map, the following is posed:

$$\nabla W^L_{N(p,q,r)} = \sum_{N=1}^{Xc}\left(\sum_{i=0}^{Xa-Ka}\sum_{j=0}^{Xb-kb}\frac{\partial E}{\partial X^L_{N\,(i,j)}}\frac{\partial X^L_{N\,(i,j)}}{\partial W^L_{N(p,q,r)}}\right)$$

$$\nabla W^L_{N(p,q,r)} = \sum_{N=1}^{Xc}\left(\sum_{i=0}^{Xa-Ka}\sum_{j=0}^{Xb-kb}\nabla X^L_{N(i,j)}\frac{\partial}{\partial W^L_{N(p,q,r)}} g\left[\sum_p\sum_q\sum_r W^L_{N(p,q,r)} * X^{L-1}_{N(i-p,j-q)} + b^L_N\right]\right)$$

$$\nabla W^L_{N(p,q,r)} = \sum_{N=1}^{Xc}\left(\sum_{i=0}^{Xa-Ka}\sum_{j=0}^{Xb-kb}\nabla X^L_{N(i,j)} \cdot d(g) * X^{L-1}_{N(i-p,j-q)}\right)$$

Convolution is an operation that turns two functions into a third one, representing a magnitude that superimposes both. When becoming a translated and inverted version, in backpropagation the weight (kernel) must rotate 180 degrees (in theory translate rows and columns) to return to the original matrix. Therefore, weight updating $W^L_{N(p,q,r)}$ is:

$$W^{L}_{N(p,q,r)actual} = W^{L}_{N(p,q,r)initial} - \eta \left[\sum_{N=1}^{Xc} \left(\sum_{i=0}^{Xa-Ka} \sum_{j=0}^{Xb-kb} \nabla X^{L}_{N(i,j)} \cdot d(g) \cdot rot\ 180\left(X^{L-1}_{N(i-p,j-q)}\right) \right) \right] \quad (43)$$

2. *Bias updating* b^{L}_{N}.

$$b^{L}_{N.actual} = b^{L}_{N.initial} - \eta\, \nabla b^{L}_{N}$$

Whereby:

$$\nabla b^{L}_{N} = \frac{\partial E}{\partial b^{L}_{N}}$$

Applying the chain rule:

$$\nabla b^{L}_{N} = \sum_{N=1}^{Xc} \left(\sum_{i=0}^{Xa-1} \sum_{j=0}^{Xb-1} \frac{\partial E}{\partial X^{L}_{N\,(i,j)}} \frac{\partial X^{L}_{N\,(i,j)}}{\partial b^{L}_{N}} \right)$$

$$\nabla b^{L}_{N} = \sum_{N=1}^{Xc} \left(\sum_{i=0}^{Xa-1} \sum_{j=0}^{Xb-1} \nabla X^{L}_{N(i,j)} \frac{\partial}{\partial b^{L}_{N}}\, g \left[\sum_{p} \sum_{q} \sum_{r} W^{L}_{N(p,q,r)} * X^{L-1}_{N(i-p,j-q)} + b^{L}_{N} \right] \right)$$

$$\nabla b^{L}_{N} = \sum_{N=1}^{Xc} \left(\sum_{i=0}^{Xa-1} \sum_{j=0}^{Xb-1} \nabla X^{L}_{N(i,j)}\, d(g) \right)$$

The bias updating is:

$$b^{L}_{N.actual} = b^{L}_{N.initial} - \eta \left[\sum_{N=1}^{Xc} \left(\sum_{i=0}^{Xa-1} \sum_{j=0}^{Xb-1} \nabla X^{L}_{N(i,j)}\, d(g) \right) \right] \quad (44)$$

3. *Updating inputs in convolutional layers* $X^{L-1}_{N(i,j,k)}$.

If it works with several convolutional layers, it is necessary to update the input $X^{L-1}_{N(i,j)}$; solving the problem of obtaining the desired output Y_i in intermediate layers for the error calculation. However, if it is the first layer, only weights and biases will be updated.

$$X^{L-1}_{N(i,j,k)actual} = X^{L-1}_{N(i,j,k)initial} - \eta\, \nabla X^{L-1}_{N(i,j,k)}$$

Whereby:

$$\nabla X_{N(i,j,k)}^{L-1} = \sum_{N=1}^{Xc}\left(\sum_{p=0}^{Ka-1}\sum_{q=0}^{Kb-1}\frac{\partial E}{\partial X_{N(i-p,j-q,k)}^{L}}\frac{\partial X_{N(i-p,j-q,k)}^{L}}{\partial X_{N(i,j,k)}^{L-1}}\right)$$

$$i = 0,1..(Xa-1);\ j = 0,1\ldots(Kb-1);\ k = N = 1,2\ldots s$$

Considering that:

$$\frac{\partial E}{\partial X_{N(i-p,j-q,k)}^{L}} = \nabla X_{N(i-p,j-q,k)}^{L} = \sum_{p}\sum_{q}\sum_{r} X_{N(i-p,j-q,k)}^{L-1} * W_{N(p,q,r)}^{L} + b_{N}^{L}$$

Replacing in the equation, it would have:

$$\nabla X_{N(i,j,k)}^{L-1} = \sum_{N=1}^{Xc}\left(\sum_{p=0}^{Ka-1}\sum_{q=0}^{Kb-1}\nabla X_{N(i-p,j-q,k)}^{L}\frac{\partial}{\partial X_{N(i,j,k)}^{L-1}}g\left[\sum_{p}\sum_{q}\sum_{r} W_{N(p,q,r)}^{L} * X_{N(i-p,j-q,k)}^{L-1} + b_{N}^{L}\right]\right)$$

$$\nabla X_{N(i,j,k)}^{L-1} = \sum_{N=1}^{Xc}\left(\sum_{p=0}^{Ka-1}\sum_{q=0}^{Kb-1}\nabla X_{N(i-p,j-q,k)}^{L}\cdot d(g) * W_{N(p,q,r)}^{L}\right)$$

When having a convolution, we must rotate $W_{N(p,q,r)}^{L}$ to $rot\ 180\ (W_{N(p,q,r)}^{L})$.

$$\nabla X_{N(i,j,k)}^{L-1} = \sum_{N=1}^{Xc}\left(\sum_{p=0}^{Ka-1}\sum_{q=0}^{Kb-1}\left[(rot\ 180\ W_{N(p,q,r)}^{L})\cdot\nabla X_{N(i-p,j-q,k)}^{L}\cdot d(g)\right]\right)$$

The input updating $\nabla X_{N(i,j,k)}^{L-1}$ is:

$$X_{N(i,j,k)actual}^{L-1} = X_{N(i,j,k)initial}^{L-1} - \eta\left[\sum_{N=1}^{Xc}\left(\sum_{p=0}^{Ka-1}\sum_{q=0}^{Kb-1}\left[(rot\ 180\ W_{N(p,q,r)}^{L})\cdot\nabla X_{N(i-p,j-q)}^{L}\cdot d(g)\right]\right)\right] \quad (45)$$

Application of mathematical modeling to an architecture CNN

To exemplify the mathematical modeling backpropagation, an architecture with two convolutional layers, two Pooling layers, a densely connected layer and an output layer was designed. The following was used: Binary cross-entropy as cost function, tanh and sigmoid as activation functions (FA), learning rate $\eta = 0.0001$ and the stochastic

descending gradient SGD as optimization function. The network aimed to classify images of motorcycles and planes (binary classification) (See Figure 2).

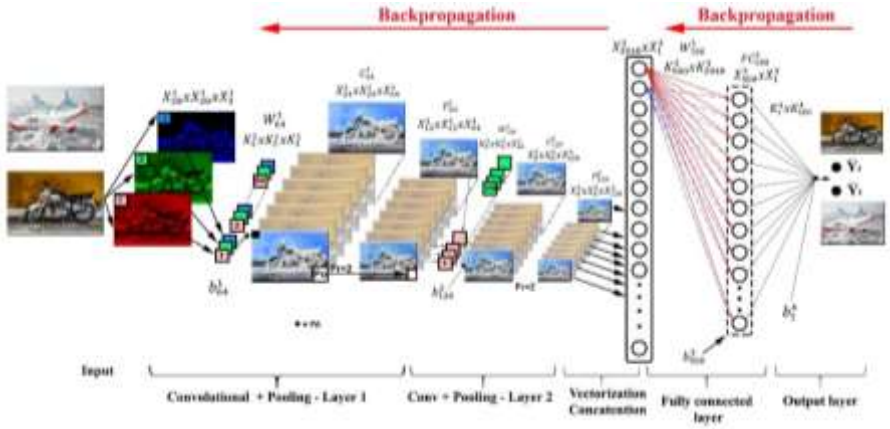


Figure 2. *Architecture of a CNN with two convolutions, two pooling, connected and an output layer.* In each phase the number of layers, filters and dimensionality (two-dimensional or three-dimensional) are indicated.

Based on equations 29,30 and 32, the simplified mathematical model of the architecture proposed in Figure 2 is represented by:

$$C = P\left[g\left(\sum_{p=1}^{K_a}\sum_{q=1}^{K_b}\sum_{r=1}^{K_c}\left[P\left[g\left(\sum_{p=1}^{K_a}\sum_{q=1}^{K_b}\sum_{r=1}^{K_c} X^{L}_{N(i-p,j-q,k)} * W^{L}_{N(p,q,r)} + b^{L}_{N}\right)\right]_{\max(2i,2j,k)}\right] * W^{L}_{N(p,q,r)} + b^{L}_{N}\right)\right]_{\max(2i,2j,k)}$$

Where:

- C = two-layer convolutional and two-layer pooling process.
- P = application of the pooling layer.
- g = activation function.

Whereby:

$$\hat{y} = sigmoid\left[\sum_{j=1}^{500} C_a \cdot W^{T}_{i,j} + b_i\right]$$

Where:

- C_a =flattening process of phase C.
- $j =$ corresponds to the number of neurons in the hidden layer.
- $\hat{y} =$ a single target (motorcycle or plane).

The step-by-step process of forward and backward propagation is presented below in sections 3.2.1 and 3.2.2.

Forward propagation

Convolution layer C_1

Considering that input images are of three channels RGB (red, green, blue) $X_c = 3$, with a dimension $X = (28x28x3)$. The first convolutional layer has a kernel =5x5x3, FA= tanh and 64 output filters. Using the equations 12 and 29, it is:

$$C^1_{N\,(i,j,k)} = g\left[\sum_{p=0}^{5-1}\sum_{q=0}^{5-1}\sum_{r=0}^{3-1} W^1_{N\,(p,q,r)} X^1_{N\,(i-p,j-q,k)} + b^1_N\right]$$

$$where:\ N = 1,2\ldots.64\ ;\ i,j = 1,2\ldots 28; k = 1,2,3$$

$$C^1_{64\,(28,28,3)} = g[z]$$

$$C^1_{64\,(28,28,3)} = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

Pooling layer P_1

Size 2x2, Pr=2 and Max-pooling. With an input of (24x24x64), getting an output of (12x12x64). Basing on the Equation 30, it is.

$$P^1_{64\,(24,24)}(\max) = max\left[\sum_{P_x=0}^{2-1}\sum_{P_y=0}^{2-1} C^1_{N[2\,i-Px,2j-Py,k]}\right] \quad ; i = 1,2\ldots.\frac{24}{2} \quad ; \quad j$$

$$= 1,2\ldots.\frac{24}{2}; s = 1,2,\ldots 64$$

Convolutional layer C_2

With a dimension $X = (12x12x64)$, a kernel=5x5x64, FA= tanh and 128 output filter. Using Equations 12 and 29.

$$C^2_{N\,(i,j,k)} = g\left[\sum_{p=0}^{5-1}\sum_{q=0}^{5-1}\sum_{r=0}^{64-1} W^1_{N\,(p,q,r)} * X^1_{N\,(i-p,j-q,k)} + b^1_N\right]$$

$$where:\ N = 1,2\ldots.128\ ;\ i,j = 1,2\ldots 12; k = 1,2\ldots.64$$

$$C^2_{128\,(12,12,64)} = g[z]$$

$$C^2_{128\,(12,12,64)} = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

Pooling layer P_2

From size 2x2, Pr=2 and Max-pooling. With an input of (8x8x128), getting an output of (4x4x128). Basing in Equation 30, it is.

$$P^2_{128\,(8,8)}(\max) = max\left[\sum_{P_x=0}^{2-1}\sum_{P_y=0}^{2-1} C^2_{s[2\,i-Px,2j-Py,k]}\right] \quad ; i = 1{,}2\ldots.\frac{8}{2} \;\; ; \;\; j$$

$$= 1{,}2\ldots.\frac{8}{2}; k = 1{,}2,\ldots 128$$

Vectorization and concatenation.

If leaving the second pooling layer the input size is X = (4x4x128), the flattening to one dimension becomes in (2048,1).

Densely connected layer C3.

The entry X is a matrix (2048,1) and the weight W (1,2048), the activation function is tanh, 2048 neurons come in and 500 out. Applying Equations 12 and 32 it is:

$$\hat{Y}_i = g\,[X \cdot W^T + b] = g[z]$$

Output layer C4.

The entry X is a matrix (500,1) and the weight W (1,500), the activation function is Sigmoid which allows to perform a binary classification, with an input of 500 and output of a neuron. Applying Equations 10 and 32.

$$\hat{Y}_i = g\left[X \cdot W^T_{i,j} + b\right] = g[z]$$

$$\hat{Y}_i = \frac{1}{1 + e^{-z}}$$

Cost function

Equation 7 of the Binary cross-entropy cost function is commonly used in binary classification problems as in this case.

$$E = -\sum_{i=1}^{n}\left[Y_i \cdot \log\left(\hat{Y}_i\right) + (1 - Y_i) \cdot \log\left(1 - \hat{Y}_i\right)\right]$$

Deriving as a function of the predicted output by the network $\hat{Y}_i$:

$$\frac{\partial E}{\partial \hat{Y}_i} = \frac{Y_i}{\hat{Y}_i} + \frac{(1 - Y_i)\,(-1)}{\left(1 - \hat{Y}_i\right)} = \frac{Y_i}{\hat{Y}_i} + \frac{(Y_i - 1)}{\left(1 - \hat{Y}_i\right)} = \frac{Y_i - Y_i\hat{Y}_i + Y_i\hat{Y}_i - \hat{Y}_i}{\hat{Y}_i\left(1 - \hat{Y}_i\right)} = \frac{Y_i - \hat{Y}_i}{\hat{Y}_i\left(1 - \hat{Y}_i\right)} \quad (46)$$

Backward propagation

Backpropagation process updates the following parameters number in each network layer (See Table 5).

Table 5. *Parameters network numbers calculation.*

Layer	Calculation	Total c/layer	Weight	Bias
Convolutional C_1	$[(5x5x3)+1]64$	4864	4800	64
Convolutional C_2	$[(5x5x64)+1]128$	204928	204800	128
Densely connected C_3	$[2048+1]500$	1024500	1024000	500
Densely connected C_4 – output	$[500+1]1$	501	500	1
Total - parameters		**1234793**	**1234100**	**693**

Densely connected layer C_4

Weigh updating $W_{1,500}^{4}$ through Equations 11, 34 and 46.

$$W_{actual} = W_{initial} - \eta\left[\frac{\partial E}{\partial \hat{Y}_i}\, d(g) \cdot X^T\right]$$

$$W_{actual} = W_{initial} - 0.0001\left[\frac{Y_i - \hat{Y}_i}{\hat{Y}_i(1-\hat{Y}_i)} g(z)[1-g(z)] \cdot X^T\right]$$

If $g(z) = \hat{Y}_i$

$$W_{i,j}^{L} = W_{initial} - 0.0001\left[\frac{Y_i - \hat{Y}_i}{\hat{Y}_i(1-\hat{Y}_i)} \hat{Y}_i[1-\hat{Y}_i] \cdot X^T\right]$$

$$W_{1,500}^{4} = W_{1,500.initial}^{4} - 0.0001\left[[Y_i - \hat{Y}_i] \cdot X^T\right] \quad ; i = 1 \; ; j = 1,2 \ldots .500$$

Bias updating b_1^4 through Equations 11, 35 and 46.

$$b_{actual} = b_{initial} - \eta \nabla e$$

$$b_i^L = b_{initial} - \eta\left[\frac{\partial E}{\partial \hat{Y}_i}\, d(g)\right]$$

$$b_i^L = b_{initial} - \eta \frac{Y_i - \hat{Y}_i}{\hat{Y}_i(1-\hat{Y}_i)} g(z)[1-g(z)]$$

$$b_i^L = b_{initial} - 0.0001\left[\frac{Y_i - \hat{Y}_i}{\hat{Y}_i(1-\hat{Y}_i)} \hat{Y}_i[1-\hat{Y}_i]\right]$$

$$b_1^4 = b_{1.initial}^{4} - 0.0001\left[[Y_i - \hat{Y}_i]\right] \; ; i = 1$$

Input updating X_{500}^{4}, replacing the former values getting in $W_{1,500}^{4}$ *and* b_1^4 using Equation 36.

$$X_{FC.actual}^{L} = X_{FC.initial}^{L} - \eta\left[\Delta e \cdot \left(W_{i,j}^{4}\right)^{T}\right]$$

$$X_{FC.actual}^{L} = \left[X_{FC.initial}^{4} - 0.0001\left(\frac{\partial E}{\partial \hat{Y}_i}\, d(g)\left(W_{i,j}^{4}\right)^{T}\right)\right] \;; j = 1,2..500$$

$$X_{Fc.actual}^{L} = \left[X_{FC.initial}^{4} - 0.0001\left(\frac{Y_i - \hat{Y}_i}{\hat{Y}_i(1-\hat{Y}_i)}\hat{Y}_i[1-\hat{Y}_i]\left(W_{i,j}^{4}\right)^{T}\right)\right] \;; i = 1\,; j = 1,2..500$$

$$X_{500}^{4} = \left[X_{500.initial}^{4} - 0.0001\left(\left(Y_i - \hat{Y}_i\right)\left(W_{i,j}^{4}\right)^{T}\right)\right] \;; i = 1 \;; j = 1,2..500$$

Densely connected layer C_3

Weigh updating $W_{500,2048}^{3}$ through Equation 37. It considers $\hat{Y}_i = X_{500}^{4}$ from C_4 layer.

$$W_{actual} = W_{initial} - \eta\left[\Delta e \cdot \left(W_{i,j}^{L}\right)^{T} d(g) X_{i,j}^{L-1}\right]$$

$$W_{actual} = W_{initial} - 0.0001\left[\left(Y_i - \hat{Y}_i\right)\left(W_{i,j}^{4}\right)^{T} d(g) \cdot X_{i,j}^{L-1}\right]$$

If $g(z) = X_{500}^{4}$

$$W_{500,2048}^{3} = W_{500,2048.initial}^{3} - 0.0001\left[\left(Y_i - \hat{Y}_i\right)\left(W_{1,500}^{4}\right)^{T} d(X_{500}^{4}) \cdot X_{i,j}^{L-1}\right]; i = 1,2..500 \;; j = 1,2\ldots.2048$$

Bias updating b_{500}^{3} through equation 38.

$$b_{actual} = b_{initial} - \eta\left[\Delta e \cdot \left(W_{i,j}^{L}\right)^{T} d(g)\right]$$

$$b_{500}^{3} = b_{500.initial}^{3} - \eta\left[\left(Y_i - \hat{Y}_i\right)\left(W_{1,500}^{4}\right)^{T} d(X_{500}^{4})\right]$$

Input updating X_{2048}^{3}, using Equation 39.

$$X_{FC.actual}^{L} = X_{FC.initial}^{L} - \eta\left[\nabla X_{FC}^{L}\, d(g)\left(W_{i,j}\right)^{T}\right]$$

$$X_{FC.actual}^{L} = X_{FC.initial}^{L} - \eta\left[\Delta e \cdot \left(W_{i,j}^{L}\right)^{T} d(g)\left(W_{i,j}\right)^{T}\right]$$

$$X_{2048}^{3} = X_{2048.initial}^{3} - \eta\left[\left(Y_i - \hat{Y}_i\right)\left(W_{1,500}^{4}\right)^{T} d(X_{500}^{4})\left(W_{i,j}\right)^{T}\right] \;; i = 1,2..500 \;; j = 1,2\ldots.2048$$

Vectorization and concatenation inverse

Through Equation 40, the flattened matrix becomes X_c filters with dimension $X_a \cdot X_b$; where $X^L_{N(i,j,k)}$ is the second pooling layer output with dimensions (4x4x128).

$$P^L_{N(i,j,k)} = \sum_{N=1}^{X_c} \left(\sum_{i=0}^{X_a-1} \sum_{j=0}^{X_b-1} X^L_{FC\,(N \cdot X_b+i)} \right)$$

$$P^2_{128(i,j)} = \sum_{s=1}^{128} \left(\sum_{i=0}^{4-1} \sum_{j=0}^{4-1} X^3_{2048\,(N \cdot X_b+i)} \right)$$

Pooling layer P_2

For pooling layer, it keeps the position (i,j) of maximum value in each region $P_o = 2x2$ from forward propagation process, which awards for each filter a value of e $X^2_{128(i,j,k)}$ (See Equation 42). The other positions will take a value of zero.

$$X^1_{128(8,8)} = \sum_{N=1}^{128} \left(\sum_{i=0}^{4-1} \sum_{j=0}^{4-1} P^2_{N(i,j,k)} = X^2_{128(i,j,k)} \right)$$

Convolutional layer C_2

Weigh updating $W^2_{128(5,5,64)}$ through Equation 43.

$$W^L_{N(p,q,r)actual} = W^L_{N(p,q,r)initial} - \eta \left[\sum_{N=1}^{Xc} \left(\sum_{i=0}^{Xa-Ka} \sum_{j=0}^{Xb-kb} \nabla X^L_{N(i,j,k)} \cdot d(g) \cdot rot\ 180\left(X^{L-1}_{N(i-p,j-q,k)}\right) \right) \right]$$

where: $P^1_{N\,(i,j,k)} = X^{L-1}_{N(i-p,j-q,k)}$, therefore:

$$W^2_{128(5,5,64)actual} = W^2_{128(5,5,64)initial} - \eta \left[\sum_{N=1}^{128} \left(\sum_{i=0}^{12-5} \sum_{j=0}^{12-5} \nabla X^2_{128(i,j,k)} \cdot d\left(X^2_{128(i,j,k)}\right) \cdot rot\ 180\left(P^1_{64\,(i,j,k)}\right) \right) \right]$$

Bias updating b^2_{128} using Equation 44.

$$b^L_{N.actual} = b^L_{N.initial} - \eta \left[\sum_{N=1}^{Xc} \left(\sum_{i=0}^{Xa-1} \sum_{j=0}^{Xb-1} \nabla X^L_{N(i,j,k)}\, d(g) \right) \right]$$

$$b^2_{128.actual} = b^2_{128.initial} - \eta\left[\sum_{N=1}^{128}\left(\sum_{i=0}^{8-1}\sum_{j=0}^{8-1} X^2_{128(i,j,k)}\, d(X^2_{128(i,j,k)})\right)\right]$$

Input updating $X^{2-1}_{64(12,12,64)}$ through Equation 45.

$$X^{L-1}_{N(i,j,k)actual} = X^{L-1}_{N(i,j,k)initial} - \eta\left[\sum_{N=1}^{Xc}\left(\sum_{p=0}^{Ka-1}\sum_{q=0}^{Kb-1}\left[\left(rot\ 180\ W^L_{N(p,q,r)}\right)\cdot \nabla X^L_{N(i-p,j-q,k)}\ \cdot d(g)\right]\right)\right]$$

$$X^{2-1}_{64(12,12,64)actual} = X^{2-1}_{64(12,12,64)initial} - \eta\left[\sum_{N=1}^{64}\left(\sum_{p=0}^{5-1}\sum_{q=0}^{5-1}\left[\left(rot\ 180\ W^2_{128(5,5,64)}\right)\cdot X^2_{128(i,j,k)}\ \cdot d(X^2_{128(i,j,k)})\right]\right)\right]$$

Pooling layer P_1

By storing the position (i,j) of the maximum value of the region $P_o = 2x2$ in the forward propagation process, each filter is assigned a value of $X^1_{64(i,j,k)}$ (See Equation 42). The other positions take a value of zero.

$$X^1_{64(24,24,64)} = \sum_{N=1}^{64}\left(\sum_{i=0}^{12-1}\sum_{j=0}^{12-1} P^1_{N(i,j,k)} = X^1_{64(i,j,k)}\right)$$

Convolutional layer C_1

Weight update $W^2_{64(5,5,3)}$ through Equation 43.

$$W^L_{N(p,q,r)actual} = W^L_{N(p,q,r)initial} - \eta\left[\sum_{N=1}^{Xc}\left(\sum_{i=0}^{Xa-Ka}\sum_{j=0}^{Xb-kb} \nabla X^L_{N(i,j,k)}\cdot d(g)\cdot rot\ 180\left(X^{L-1}_{N(i-p,j-q,k)}\right)\right)\right]$$

where: initial matrix $X^1_{3\ (28,28,3)} = X^{L-1}_{N(i-p,j-q,k)}$, therefore:

$$W^1_{64(5,5,3)actual} = W^1_{64(5,5,3)initial} - \eta\left[\sum_{N=1}^{64}\left(\sum_{i=0}^{24-5}\sum_{j=0}^{24-5} X^1_{64(i,j,k)}\cdot d\left(X^1_{64(i,j,k)}\right)\cdot rot\ 180\left(X^1_{3\ (28,28,3)}\right)\right)\right]$$

Bias update b^1_{64} using Equation 44.

$$b^{L}_{N.actual} = b^{L}_{N.initial} - \eta \left[\sum_{N=1}^{Xc} \left(\sum_{i=0}^{Xa-1} \sum_{j=0}^{Xb-1} \nabla X^{L}_{N(i,j,k)} \, d(g) \right) \right]$$

$$b^{1}_{64.actual} = b^{1}_{64.initial} - \eta \left[\sum_{N=1}^{64} \left(\sum_{i=0}^{24-1} \sum_{j=0}^{24-1} X^{1}_{64(i,j,k)} \, d(X^{1}_{64(i,j,k)}) \right) \right]$$

1. DISCUSSION

The development of mathematical modeling in a convolutional neural network (CNN), allowed to demystify and solve several authors' problem such as Buhrmester (Buhrmester et al., 2019), Sturm (Sturm et al., 2016) y Baselli (Baselli et al., 2020). They describe the process of a CNN as a black box; forcing researchers in the absence of other studies to work only under their general equations and limited information.

The detailed mathematical process of a CNN confirms Buhrmester's idea (Buhrmester et al., 2019) outlined in his article "Analysis of Explainers of Black Box Deep Neural Networks for Computer Vision: A Survey". It emphasizes the importance of a complete knowledge of the mathematical process that a CNN network pursues on effective optimization.

The modeling of the convolution, pooling and densely connected layers for three-dimensional images in the forward propagation process has several modifications for the case (color image). However, it agrees essentially with the proposed final equations by Dumoulin (Dumoulin & Visin, 2018), Kuo (Kuo, 2016) and Koushik (Koushik, 2016) used in the case of a two-dimensional image (black and white). In addition, a contribution was made to the updating of the network parameters (backpropagation in the three-dimensional case) using the stochastic gradient descent optimization equation (SGD) posed by Bottou (Bottou, 2012).

CONCLUSIONS AND SOCIAL IMPLICATIONS

- From the exhaustive bibliographic review, it is concluded that there are no articles that present a detailed step-by-step mathematical process of forward and backward propagation in a CNN for three-dimensional images (color), even for the two-dimensional case, the information is concise and scarce, preventing its understanding by researchers who do not master the mathematical area.

- This work allows reaching readers with different levels of knowledge (beginners, intermediate and specialists), providing consistent, effective and understandable information on the process of one of the algorithms with the greatest impact, such as convolutional neural networks. It becomes a relevant contribution to the scientific community that allows identifying intermediate processes and in turn posing optimizations with solid bases in relation to its mathematical model.
- The exemplification of the modeled formulas for the forward and backward propagation process allowed to demonstrate its easy application in any architecture used by the researchers. Additionally, it allows to verify the amount of learning parameters of the network; understanding the complexity of the model and the differences in execution times either by the CPU or GPU.
- The development of tables with the compilation of the main cost, activation and optimization functions, facilitates the reader the modeling of different architectures with specific functions depending on the problem to be solved; without the need to modify the variables identification posed by various authors according to the models posed in this article.
-

REFERENCES

Asghar, M. Z., Habib, A., Habib, A., Khan, A., Ali, R., & Khattak, A. (2019). Exploring deep neural networks for rumor detection. *Journal of Ambient Intelligence and Humanized Computing*, 4315–4333. https://doi.org/10.1007/s12652-019-01527-4

Baselli, G., Codari, M., & Sardanelli, F. (2020). Opening the black box of machine learning in radiology: Can the proximity of annotated cases be a way? *European Radiology Experimental*, *4*(1), 30. https://doi.org/10.1186/s41747-020-00159-0

Berzal, F. (2018). Redes Neuronales & Deep Learning. In *Redes Neuronales & Deep Learning* (pp. 194–200).

Bottou, L. (2012). Stochastic Gradient Descent Tricks. In G. Montavon, G. B. Orr, & K.-R. Müller (Eds.), *Neural Networks: Tricks of the Trade* (Vol. 7700, pp. 421–436). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-35289-8_25

Buhrmester, V., Münch, D., & Arens, M. (2019). Analysis of Explainers of Black Box Deep Neural Networks for Computer Vision: A Survey. *ArXiv:1911.12116 [Cs]*. http://arxiv.org/abs/1911.12116

Dhillon, A., & Verma, G. K. (2020). Convolutional neural network: A review of models, methodologies and applications to object detection. *Progress in Artificial Intelligence*, *9*(2), 85–112. https://doi.org/10.1007/s13748-019-00203-0

Dumoulin, V., & Visin, F. (2018). A guide to convolution arithmetic for deep learning. *ArXiv:1603.07285 [Cs, Stat]*. http://arxiv.org/abs/1603.07285

Ghosh, A., & Kandasamy, D. (2020). Interpretable Artificial Intelligence: Why and When. *American Journal of Roentgenology*, *214*(5), 1137–1138. https://doi.org/10.2214/AJR.19.22145

Glorot, X., Bordes, A., & Bengio, Y. (2011). *Deep Sparse Rectifier Neural Networks*. *15*, 9.

Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., Liu, T., Wang, X., Wang, G., Cai, J., & Chen, T. (2018). Recent advances in convolutional neural networks. *Pattern Recognition*, *77*, 354–377. https://doi.org/10.1016/j.patcog.2017.10.013

Haykin, S. (2009). Neural Networks and Learning Machines. In *Neural Networks and Learning Machines* (Third, pp. 478–481). Pearson-Prentice Hall.

Janocha, K., & Czarnecki, W. M. (2017). On Loss Functions for Deep Neural Networks in Classification. *ArXiv:1702.05659*. http://arxiv.org/abs/1702.05659

Kim, Y. (2014). Convolutional Neural Networks for Sentence Classification. *ArXiv:1408.5882 [Cs]*. http://arxiv.org/abs/1408.5882

Koushik, J. (2016). Understanding Convolutional Neural Networks. *ArXiv:1605.09081 [Stat]*. http://arxiv.org/abs/1605.09081

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, *60*(6), 84–90. https://doi.org/10.1145/3065386

Kullback, S., & Leibler, R. A. (1951). On Information and Sufficiency. *Annals of Mathematical Statistics*, *22*(1), 79–86. https://doi.org/10.1214/aoms/1177729694

Kuo, C.-C. J. (2016). Understanding convolutional neural networks with a mathematical model. *Journal of Visual Communication and Image Representation*, *41*, 406–413. https://doi.org/10.1016/j.jvcir.2016.11.003

Lazzeri, F. (2020, May 7). *Selección de un algoritmo de Machine Learning*. Azzure Machine Learning. https://docs.microsoft.com/es-es/azure/machine-learning/how-to-select-algorithms

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436–444. https://doi.org/10.1038/nature14539

Lee, H., Grosse, R., Ranganath, R., & Ng, A. Y. (2009). Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. *Proceedings of the 26th Annual International Conference on Machine Learning - ICML '09*, 1–8. https://doi.org/10.1145/1553374.1553453

Liu, T., Fang, S., Zhao, Y., Wang, P., & Zhang, J. (2015). Implementation of Training Convolutional Neural Networks. *ArXiv:1506.01195-Computer Vision and Pattern Recognition*, 10. arXiv:1506.01195.

Lopez-Pinaya, W., Vieira, S., Garcia-Dias, R., & Mechelli, A. (2020). Chapter 10—Convolutional neural networks. In A. Mechelli & S. Vieira (Eds.), *Machine Learning* (pp. 173–191). Academic Press. https://doi.org/10.1016/B978-0-12-815739-8.00010-9

Martineau, M., Raveaux, R., Conte, D., & Venturini, G. (2020). A Convolutional Neural Network into graph space. *ArXiv:2002.09285 [Cs]*. http://arxiv.org/abs/2002.09285

Minaee, S., Kalchbrenner, N., Cambria, E., Nikzad, N., Chenaghlu, M., & Gao, J. (2020). Deep Learning Based Text Classification: A Comprehensive Review. *ArXiv:2004.03705*. http://arxiv.org/abs/2004.03705

Namikawa, K., Hirasawa, T., Yoshio, T., Fujisaki, J., Ozawa, T., Ishihara, S., Aoki, T., Yamada, A., Koike, K., Suzuki, H., & Tada, T. (2020). Utilizing artificial intelligence in endoscopy: A clinician's guide. *Expert Review of Gastroenterology & Hepatology*, 1–18. https://doi.org/10.1080/17474124.2020.1779058

Parmar, R. (2018, September 2). *Common Loss functions in machine learning*. Medium. https://towardsdatascience.com/common-loss-functions-in-machine-learning-46af0ffc4d23

Perconti, P., & Plebe, A. (2020). Deep learning and cognitive science. *Cognition*, *203*, 104365. https://doi.org/10.1016/j.cognition.2020.104365

Pesapane, F., Tantrige, P., Patella, F., Biondetti, P., Nicosia, L., Ianniello, A., Rossi, U. G., Carrafiello, G., & Ierardi, A. M. (2020). Myths and facts about artificial intelligence: Why machine- and deep-learning will not replace interventional radiologists. *Medical Oncology*, *37*(5), 40. https://doi.org/10.1007/s12032-020-01368-8

Rao, G. A., Syamala, K., Kishore, P. V. V., & Sastry, A. S. C. S. (2018). Deep convolutional neural networks for sign language recognition. *2018 Conference on Signal Processing And Communication Engineering Systems (SPACES)*, 194–197. https://doi.org/10.1109/SPACES.2018.8316344

Rawat, W., & Wang, Z. (2017). Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review. *Neural Computation*, *29*(9), 2352–2449. https://doi.org/10.1162/neco_a_00990

Ruder, S. (2017). An overview of gradient descent optimization algorithms. *ArXiv:1609.04747 [Cs]*. http://arxiv.org/abs/1609.04747

Sturm, I., Lapuschkin, S., Samek, W., & Müller, K.-R. (2016). Interpretable deep neural networks for single-trial EEG classification. *Journal of Neuroscience Methods*, *274*, 141–145. https://doi.org/10.1016/j.jneumeth.2016.10.008

Van Houdt, G., Mosquera, C., & Napoles, G. (2020). A review on the long short-term memory model. *Artificial Intelligence Review*. https://doi.org/10.1007/s10462-020-09838-1

Wadawadagi, R., & Pagi, V. (2020). Sentiment analysis with deep neural networks: Comparative study and performance assessment. *Artificial Intelligence Review*. https://doi.org/10.1007/s10462-020-09845-2

Wu, J. (2017). *Introduction to Convolutional Neural Networks*. 2–28. https://cs.nju.edu.cn/wujx/paper/CNN.pdf

Zhang, L., Wang, S., & Liu, B. (2018). Deep learning for sentiment analysis: A survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, *8*(4). https://doi.org/10.1002/widm.1253