



Descriptif:

Backpropagation (Rétropropagation)

Encadrant: **Hugo Bolloré**

Date: 03/01/2022

Problématique

Le but de cette partie est la description des étapes de rétropropagation (Backpropagation) utilisée dans les classes *Convolution_layer*, *Pooling_layer* et *Softmax_layer* du project Réseau de Neurones Convolutifs.

BackpropagationGeneralités et définitions:**1. Définition de la Rétropropagaion (Backpropagation):**

la rétropropagation du gradient est utilisée pour entraîner un réseau de neurones. Elle met à jour les poids de chaque neurone, en allant de la dernière couche vers la première [1], comme illustré dans la figure (1).

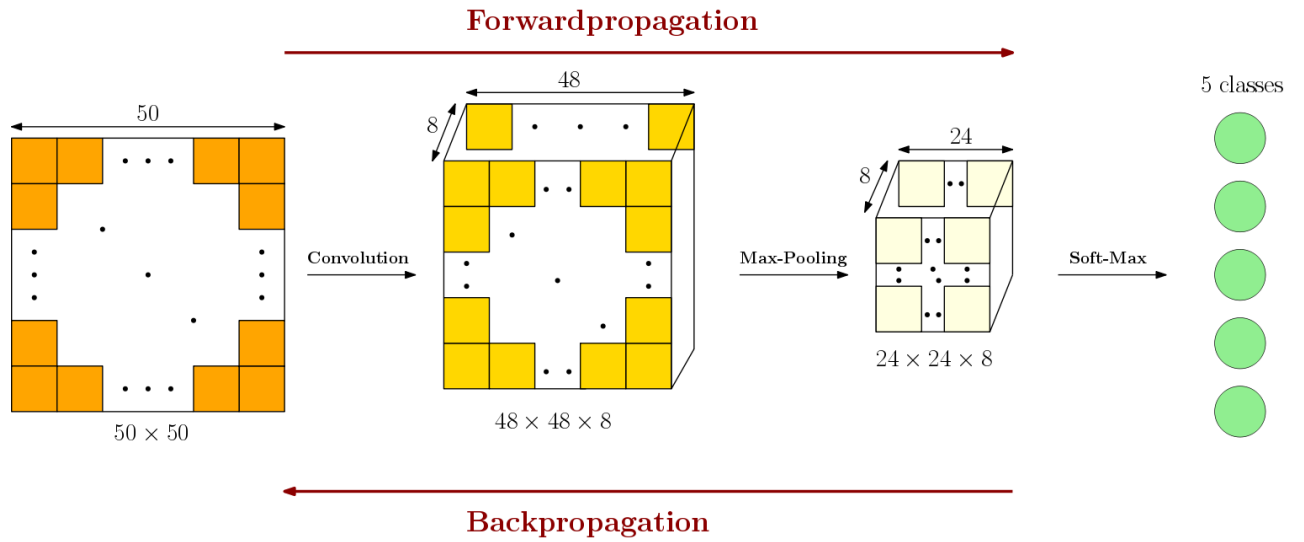


Figure 1: Forward-/ Back-propagation du réseau de neurones utilisé.

2. Rétropropagaion pour la couche Softmax:

La couche de Softmax permet de quantifier les prédictions d'appartenance d'une image donnée en entrée à une classe c parmi les 5 classes de sortie. On introduit ainsi la fonction de perte "**Cross-Entropy Loss**", caractérisée par l'équation dans [2]:

$$L_{cross-entropy}(\hat{y}, y) = - \sum_{i=1}^{n_{classes}} y_i \times \log(\hat{y}_i) \quad (1)$$

avec:

$$\begin{cases} n_{classes} : & \text{le nombre de classes,} \\ y_i & \text{la probabilité de choisir une classe (0 ou 1),} \\ \hat{y}_i & \text{la probabilité prédite pour la classe } i \text{ (par exemple 0.6).} \end{cases}$$

On se place dans le cas où $i = c$, la classe correcte qui doit être choisie. Alors $y_i = 0$ pour tous, sauf la classe correcte. Soit dans ce cas la fonction de perte réduite suivante pour $i = c$:

$$L_{cross-entropy}(\hat{y}_c) = -\log(\hat{y}_c) \quad (2)$$

où c est la classe correcte et \hat{y}_c est la probabilité prédite pour la classe c .

Le cas idéal sera pour la probabilité $\hat{y}_c = 1$, la fonction de perte $L_{cross-entropy}(\hat{y}_c)$ sera alors nulle.

La phase d'apprentissage d'un réseau de neurones se compose de deux phases:

- Phase "Forward-propagation", où l'entrée passe par les couches du réseau,
- Phase "Back-propagation", où les gradients sont rétropropagés pour mettre à jour les poids.

En se passant d'une couche à une autre dans la phase de "Forward-propagation", chaque couche cache ses données, qui seront utilisés dans la phase "Back-propagation". Ainsi, cette phase doit obligatoirement être précédée par une phase "Forward-propagation".

Par conséquent, au cours de la phase "Back-propagation", on aura:

- Chaque couche reçoit le gradient de perte par rapport à ses sorties $\frac{\partial L}{\partial out}$,
- Chaque couche renvoie le gradient de perte par rapport à ses entrées $\frac{\partial L}{\partial in}$.

On dérive l'équation (2) par rapport aux sorties de la couche softmax *out*, qui seront ses entrées dans la phase "Back-propagation". La couche softmax représente un vecteur 1D regroupant les 5 classes de fleurs.

Soit:

$$\frac{\partial L}{\partial out(i)} = \begin{cases} 0 & \text{si } i \neq c \\ \frac{-1}{y_i} & \text{si } i = c \end{cases} \quad (3)$$

Couches cachées:

On se propose d'utiliser 3 couches cachées, comme illustre dans la figure (2):

- L'entrée avant d'être aplatie (before flattening), qui représente dans notre cas un volume de dimension $(24 \times 24 \times 8)$,
- L'entrée après être aplatie (after flattening), qui sera dans ce cas un vecteur 1D de dimension (4608×1) ,
- Les valeurs passées à la fonction d'activation softmax.

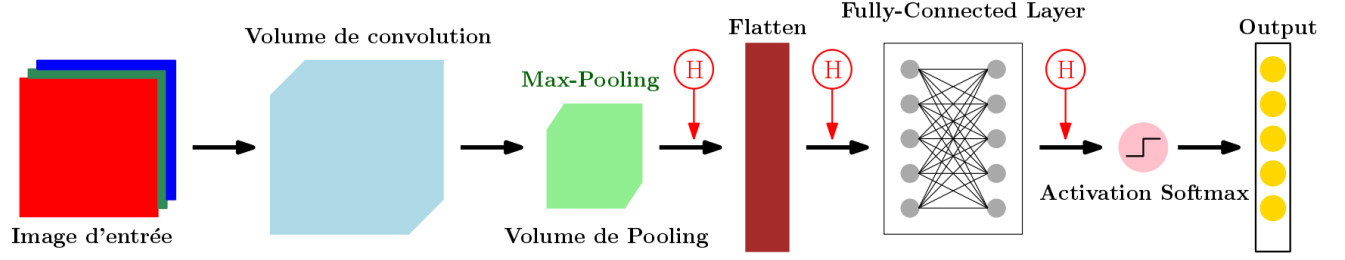


Figure 2: Représentation de couches cachées.

On pose t_i le total des classes dans le réseau de neurones. Cherchons maintenant le gradient $\frac{\partial out(c)}{\partial t(i)}$, où c est la classe correcte choisie.

Comme la sortie correspondent à celle de la fonction softmax, alors on a d'après [3]:

$$out(c) = \frac{\exp(t_c)}{\sum_i \exp(t_i)} \quad (4)$$

Cette équation indique la probabilité de choisir la classe c parmi la totalité des classes t_i .

Pour simplifier, on pose dans la suite:

$$Sum = \sum_i \exp(t_i) \quad (5)$$

Alors, l'équation (4) devient:

$$out(c) = \frac{\exp(t_c)}{Sum} \quad (6)$$

On distingue deux cas:

- Cas où $i \neq c$: Comme $out(c)$ dépend de Sum , alors en utilisant la règle de la chaîne (Appendix), on aura:

$$\frac{\partial out(c)}{\partial t_i} = \frac{\partial out(c)}{\partial Sum} \times \frac{\partial Sum}{\partial t_i} = \frac{-\exp(t_c)}{Sum^2} \times \exp(t_i) \quad (7)$$

- Cas où $i = c$:

En utilisant la règle de quotient, on aura:

$$\frac{\partial out(c)}{\partial t_c} = \frac{\partial}{\partial t_c} \left(\frac{\exp(t_c)}{Sum} \right) = \frac{\exp(t_c) \times Sum - \exp(t_c) \times \frac{\partial Sum}{\partial t_c}}{Sum^2} \quad (8)$$

Notant que $\frac{\partial Sum}{\partial t_c}$ est nulle, sauf dans le cas où $t_i = t_c$, l'équation (8) est réduite à:

$$\frac{\partial out(c)}{\partial t_c} = \frac{\exp(t_c) \times (Sum - \exp(t_c))}{Sum^2} \quad (9)$$

Maintenant, on cherche à calculer les 3 gradients de perte:

- Le gradient de poids (Weight Gradient) $\frac{\partial L}{\partial W}$ pour mettre à jour les poids,
- Le gradient de Biases (Biases gradient) $\frac{\partial L}{\partial b}$ pour mettre à jour les Biases,
- Le gradient par rapport aux entrées (Input gradient) $\frac{\partial L}{\partial input}$ pour la méthode de rétropropagation, qui va être utilisée dans la couche suivante.

On introduit l'équation depuis [4], également illustré dans la figure (3):

$$T(sortie) = W \times input + b \quad (10)$$

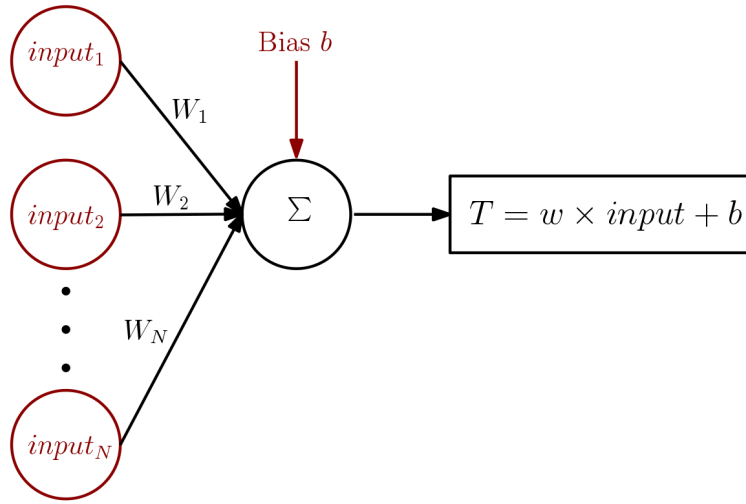


Figure 3: Equation reliant les Poids, les Entrées et les Biases.

On dérive l'équation (10) par rapport à W , $input$ et b . On obtient:

$$\begin{cases} \frac{\partial T}{\partial W} = input \\ \frac{\partial T}{\partial b} = 1 \\ \frac{\partial T}{\partial input} = W \end{cases} \quad (11)$$

En utilisant la règle de la chaîne (Appendix), on aura:

$$\begin{cases} \frac{\partial L}{\partial W} = \frac{\partial L}{\partial out} \times \frac{\partial out}{\partial T} \times \frac{\partial T}{\partial W} \\ \frac{\partial L}{\partial b} = \frac{\partial L}{\partial out} \times \frac{\partial out}{\partial T} \times \frac{\partial T}{\partial b} \\ \frac{\partial L}{\partial input} = \frac{\partial L}{\partial out} \times \frac{\partial out}{\partial T} \times \frac{\partial T}{\partial input} \end{cases} \quad (12)$$

3. Rétropropagaion pour la couche Max-Pooling:

Comme la couche de Max-Pooling n'a pas de poids, on ne peut pas faire l'apprentissage. La phase de "Backpropagation" permet de calculer les gradients.

La couche de Max-Pooling permet de réduire le volume obtenue depuis la couche de convolution à la moitié (comme expliqué précédemment). Il s'agit dans ce cas de la phase "Forwardpropagation".

Contrairement au "Forwardpropagation", la phase de "Backpropagation" permet de doubler la longueur et largeur du gradient de perte.

On considère les illustrations suivantes pour mieux comprendre la phase de "Backpropagation" dans la couche de Max-Pooling.

Soit la matrice 4×4 et la matrice de Max-Pooling illustrées dans la figure (4) dans le cas de la phase de "Forwardpropagation".

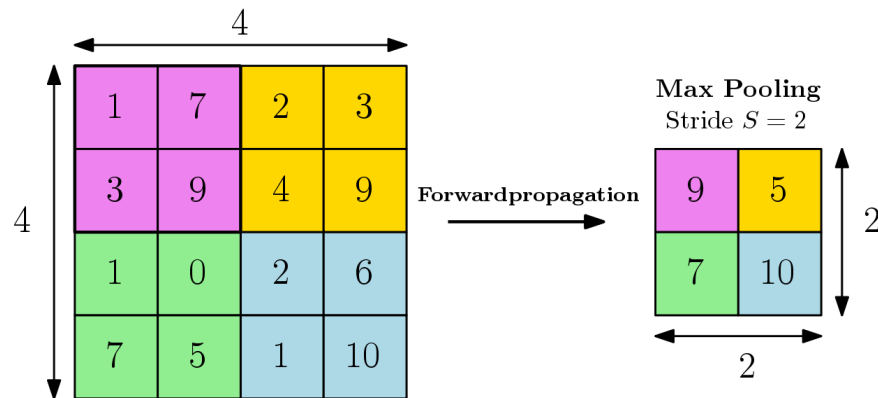


Figure 4: Matrice de Max-Pooling dans la phase de "Forwardpropagation".

Supposant maintenant qu'on dispose de la matrice 2×2 contenant les dérivées de L par rapport aux dérivées de $inputs$. Dans la phase de "Backpropagation", on a besoin de créer une matrice 4×4 . La position de la dérivée reste la même que celle où on a trouvé la valeur maximale car on a $\frac{\partial L}{\partial input} = 1$, donc $\frac{\partial L}{\partial input} = \frac{\partial L}{\partial output}$. Les autres pixels auront zéro comme valeur, car $\frac{\partial L}{\partial input} = 0$ pour les valeurs non maximales.

En conclusion, on copie le gradient de $\frac{\partial L}{\partial output}$ à $\frac{\partial L}{\partial input}$ dans le cas d'une valeur maximale.

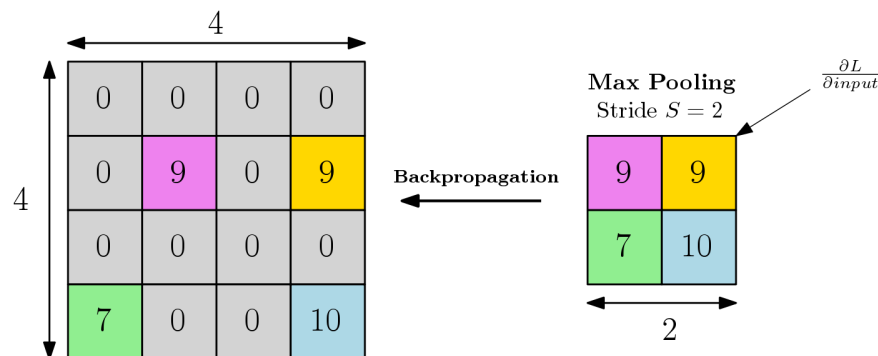


Figure 5: Matrice de Convolution obtenue dans la phase de "Backpropagation".

4. Rétropropagaion pour la couche de Convolution:

La rétropropagation est une étape cruciale dans la couche de convolution, car elle permet de mettre à jour les poids des filtres initialisés au debut par des valeurs aléatoires.

On cherche dans cette étape à trouver la dérivée $\frac{\partial L}{\partial W}$, qu'on peut la décomposer en utilisant la règle de chaîne (Appendix) de la manière suivante:

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial out} \times \frac{\partial out}{\partial W} \quad (13)$$

Comme on connaît déjà la valeur de la dérivée $\frac{\partial L}{\partial out}$, il nous reste que déterminer $\frac{\partial out}{\partial W}$. A ce stade, on se propose la question suivante: Comment le changement des valeurs de poids peut affecter la perte?

Pour y répondre, on se propose de deux cas:

- Cas trivial: On prend comme exemple une image d'entrée 3×3 et un filtre de dimension 3×3 avec des poids tous nuls. On aura dans ce cas une sortie de dimension 1×1 nulle de la couche de convolution (cas de figure (6)),

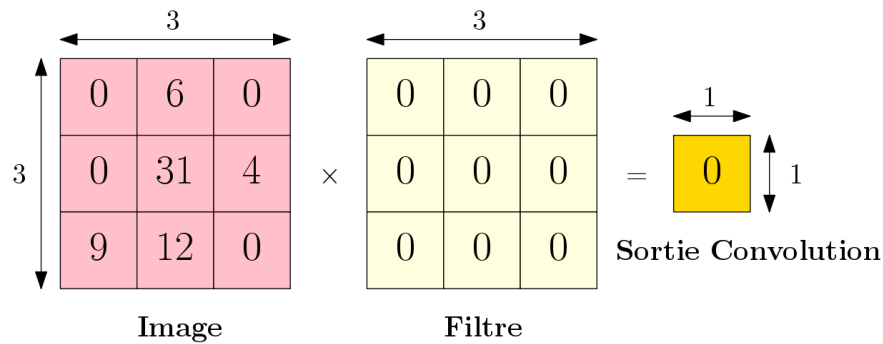


Figure 6: Convolution pour des poids de filtre nulles.

- Cas où le poids placé au centre du filtre est non nul: On reprend le même exemple, et on change la valeur du poids placée au centre du filtre à 1. La valeur centrale de la matrice de convolution sera égale à 31 comme illustré dans la figure (7). Par conséquent, la dérivée $\frac{\partial L}{\partial W}$ obtiendra l'image d'entrée, i.e., $output_image(i, j) = convolution(Image, filter)$

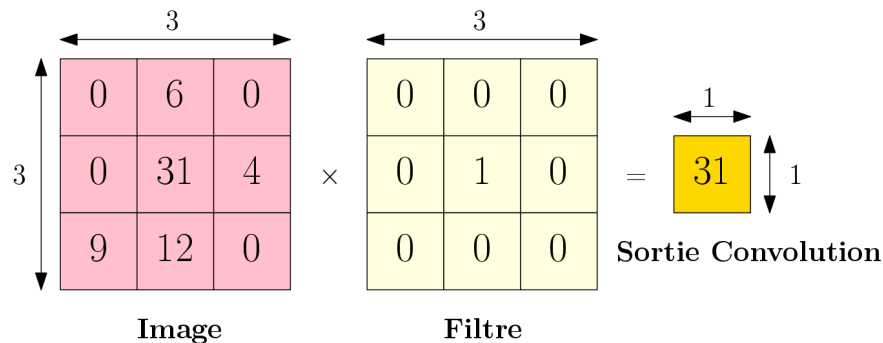


Figure 7: Convolution pour un poid au centre du filtre non nul.

Appendix:

On considère un noeud z d'un réseau de neurones [2].

Règle de la chaîne (Chain Rule):

Il s'agit d'une règle de dérivation, qui permet de calculer les dérivées des fonctions composées.

- On pose dans le cas où z ne dépend que de a :

$$\begin{cases} a(t) = \text{fonction}(t) & \text{différentiable en } t \\ z(a(t)) = \text{fonction}(a(t)) & \text{différentiable en } a \end{cases} \quad (14)$$

Alors la dérivée de z par rapport à t dans le cas d'une dépendance unique de Z en t est:

$$\frac{\partial z}{\partial t} = \frac{\partial z}{\partial a} \times \frac{\partial a}{\partial t} \quad (15)$$

Cette procédure est illustrée dans la figure (1).

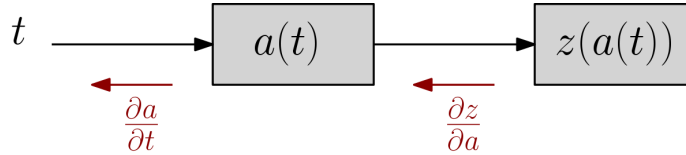


Figure 8: Règle de la chaîne appliquée à une fonction composée de dépendance simple.

- On pose dans le cas où z dépend que de a_1 et a_2 :

$$\begin{cases} a_1(t), a_2(t) = \text{fonction}(t) & \text{différentiable en } t \\ z(a_1(t), a_2(t)) = \text{fonction}(a_1(t), a_2(t)) & \text{différentiable en } a \end{cases} \quad (16)$$

Alors la dérivée de z par rapport à t dans le cas d'une dépendance unique de Z en t est:

$$\frac{\partial z}{\partial t} = \frac{\partial z}{\partial a_1} \times \frac{\partial a_1}{\partial t} + \frac{\partial z}{\partial a_2} \times \frac{\partial a_2}{\partial t} \quad (17)$$

Cette procédure est illustrée dans la figure (2).

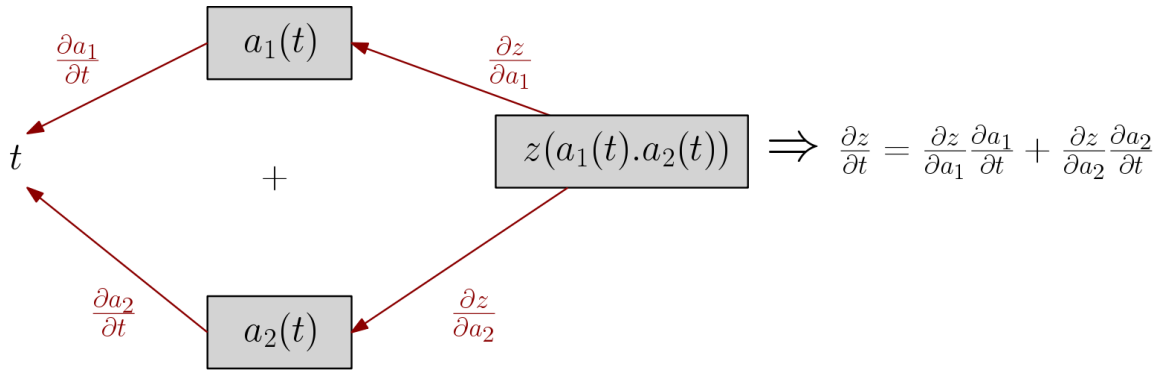


Figure 9: Règle de la chaîne appliquée à une fonction composée de dépendance multiple.

Gradient:

D'après [2], on définit:

- **Gradient Amont (Upstream gradient):** C'est le gradient que le noeud z reçoit lors de la rétropropagation (Backpropagation),
- **Gradients locaux (Local gradients):** C'est les gradients calculés par rapport aux entrées de z ,
- **Gradients Aval (Downstream gradients):** C'est le produit entre le Gradient Amont et Gradients Locaux .

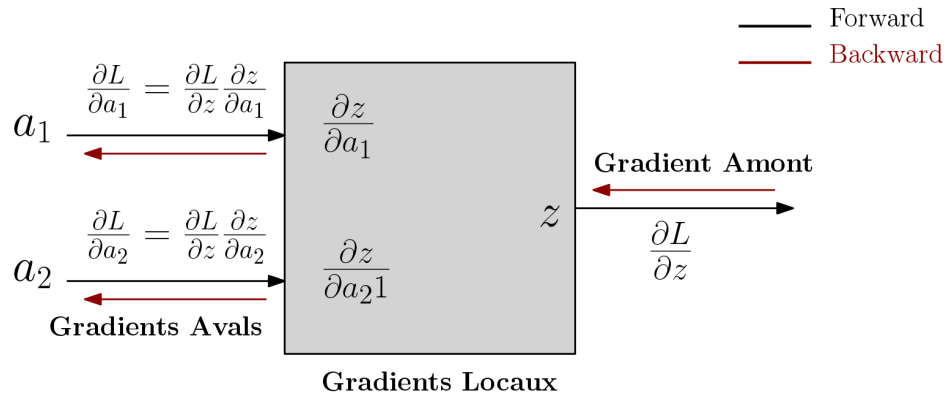


Figure 10: Formes de Gradients.

Règle de Quotient (Quotient Rule):

Soit deux fonctions f et g dépendantes de t . Alors, on a la dérivée du quotient suivant:

$$\frac{\partial}{\partial t} \left(\frac{f(t)}{g(t)} \right) = \frac{f'(t) \times g(t) - f(t) \times g'(t)}{g(t)^2} \quad (18)$$

References:

[1] "Rétropropagation du gradient", Wikipedia, 2021, "https://fr.wikipedia.org/wiki/R%C3%A9tropropagation_du_gradient".

[2] "BA Friendly Introduction To Cross-Entropy For Machine learning", Pianalytix, 2021, "<https://pianalytix.com/a-friendly-introduction-to-cross-entropy-for-machine-learning/>".

[3] "Convolutional Neural Networks (CNN): Softmax & Cross-Entropy", Super Data Science Team, 2018, "<https://www.andreaperlato.com/aipost/cnn-and-softmax/>".

[4] "Weights and Biases", AI Wiki, 2021, "<https://docs.paperspace.com/machine-learning/wiki/weights-and-biases>".

[1] "Multi-Layer Neural Networks with Sigmoid Function", Nahua Kang, Towards Data Science, 2017, "<https://towardsdatascience.com/multi-layer-neural-networks-with-sigmoid-function-deep-learning-for-rookies-2-bf464f09eb7f>".

[2] "Backpropagation in Fully Convolutional Networks", Giuseppe Pio Cannata, Towards Data Science, 2021, "<https://towardsdatascience.com/backpropagation-in-fully-convolutional-networks-fcns-1a13b75fb56a>".