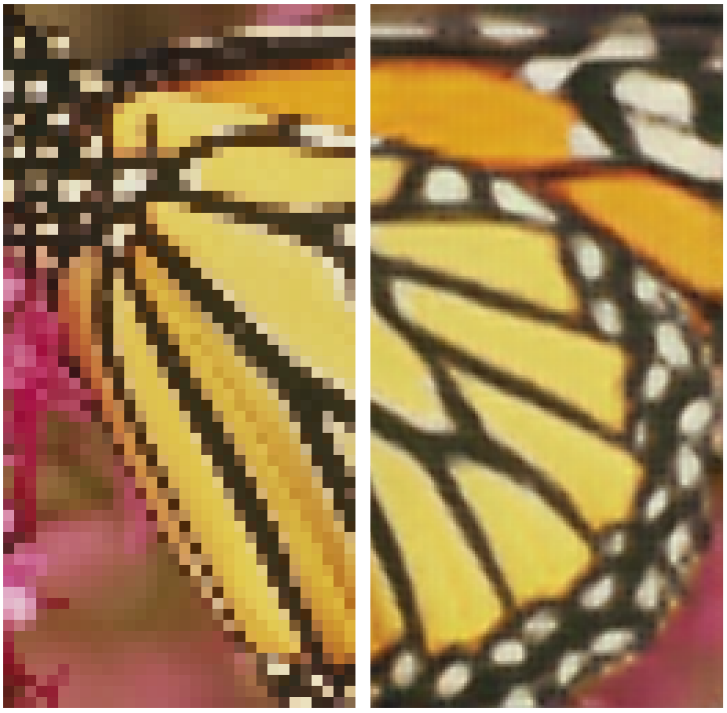
Universitat Autònoma de Barcelona

TREBALL DE FI DE GRAU
Grau en Matemàtiques

SUPER-RESOLUCIÓ AMB DEEP LEARNING

Febrer 2022



Autor: Marc Casals i Salvador
Tutor: Roger Borràs Amoraga
NIU: 1460924

Agraïments

La realització d'aquest treball no haguera estat possible sense el compromís del meu tutor, en Roger Borràs, que ha guiat els meus passos pel camí de l'aprenentatge. Cal donar les gràcies a la meva família i als meus amics pel suport incondicional que sempre se m'ha donat, en especial a la Clara per la seva exhaustiva lectura de la meva obra.

"Learning is forgetting the details as much as it is remembering the important parts."

— Pedro Domingos

Índex

1 Introducció **1**
- 1.1 Motivació . . . 1
- 1.2 Objectius . . . 1
- 1.3 Metodologia . . . 2

2 Xarxes neuronals **3**
- 2.1 Definició . . . 3
- 2.2 Funcionament . . . 4
  - 2.2.1 Funció d'activació . . . 5
  - 2.2.2 Funció de Pèrdua . . . 7
  - 2.2.3 Entrenament . . . 7
  - 2.2.4 Gradient Descent . . . 7
  - 2.2.5 Backpropagation . . . 9
- 2.3 Aspectes tècnics de les xarxes neuronals . . . 9

3 Xarxes convolucionals **14**
- 3.1 Motivació . . . 14
- 3.2 La funció de convolució . . . 15
- 3.3 Funcionament de les xarxes convolucionals . . . 16
  - 3.3.1 Padding . . . 17
  - 3.3.2 Strides . . . 17
  - 3.3.3 Maxpooling . . . 18

4 Super-Resolució **19**
- 4.1 FSRCNN . . . 20
  - 4.1.1 Introducció . . . 20
  - 4.1.2 Estructura de la xarxa . . . 21
  - 4.1.3 Entrenament amb la llibreria MNIST . . . 23
  - 4.1.4 Entrenament amb la llibreria BSDS300 . . . 24
  - 4.1.5 Llibreria *Flickr* . . . 26

5 Conclusions **30**

Abstract

En aquest treball es fa una introducció a les xarxes neuronals denses i les xarxes neuronals convolucionals, exposant, també, els principals problemes que les concerneixen i els diferents mecanismes que són essencials pel seu funcionament.

L'objectiu del treball és dissenyar una xarxa neuronal convolucional que sigui capaç de realitzar la super-resolució, és a dir, que sigui capaç d'augmentar la resolució d'una imatge donada. Per tal d'aconseguir-ho, s'hauran d'aplicar els diferents conceptes estudiats anteriorment i, seguint l'estructura dissenyada per Chao Dong *et al.*(1), crear i entrenar una xarxa. Posteriorment, s'analitzaran els diferents resultats obtinguts i s'aplicaran una sèrie de millores que conduiran a la configuració de la xarxa definitiva que resol el problema completament. Finalment, amb diverses modificacions de la xarxa anterior, es provarà d'acomplir aquesta mateixa tasca, però augmentant la resolució de les imatges quatre cops i mostrant els resultats assolits.

En el meu GitHub podeu trobar el codi emprat i els diferents resultats obtinguts.

Capítol 1

Introducció

1.1 Motivació

En els últims anys la intel·ligència artificial ha estat un tema recurrent entre els diversos mitjans de comunicació. El meu primer contacte amb ella va ser quan estava de voluntari en les activitats dels dissabtes de les matemàtiques. En una d'aquestes activitats es feia una visita al Centre de Visió per Computadors de la UAB, on ens van mostrar els diferents projectes que estaven duent a terme. Entre ells es trobava l'automòbil autònom, diversos algoritmes capaços de determinar la quantitat de persones que hi ha en una fotografia i, fins i tot, un programa capaç de desxifrar diferents mostres de textos antics. En aquell moment estava cursant l'assignatura de mètodes numèrics i ja havia adquirit diversos coneixements del llenguatge C i de la programació en general. Però, com que les explicacions donades en aquella activitat no van ser de caire tècnic, la metodologia que s'havia implementat per aconseguir tot allò m'era totalment desconeguda.

En aquella època, era incapaç de concebre com era possible dur a terme totes aquelles tasques amb l'única programació que coneixia, la programació clàssica. Un fet tan senzill com reconèixer la cara d'una persona és totalment impossible de programar, ja que s'hauria d'implementar una infinitat de condicions lògiques que permetessin determinar la disposició dels píxels que la formen. A més, no seria extrapolable a altres imatges de la mateixa persona, perquè en la realització d'una altra fotografia, per molt similar que fos a la primera, canviaria la col·locació dels píxels i la relació que tenen entre ells.

He trobat en el Treball de Fi de Grau l'oportunitat perfecta per començar a aprendre aquestes tècniques, tant noves com fascinants, que, mitjançant algorismes de *Deep Learning*, permeten fer tasques anteriorment inimaginables. Personalment, crec que l'aplicació d'aquestes tècniques a la visió per computadors ho fa encara més interessant, pel fet que els resultats són encara més tangibles i impressionants.

1.2 Objectius

L'objectiu del treball és comprendre el funcionament de les xarxes neuronals i de les xarxes convolucionals amb tots els elements que les conformen. Per aconseguir-ho, intentarem resoldre el problema de la super-resolució. Aquest problema consisteix a augmentar la resolució d'una imatge (és a dir, la quantitat de píxels que la conformen) usant una xarxa convolucional. Així doncs, buscarem quina és l'estructura de la xarxa i quin és el conjunt de dades que genera els millors resultats.

1.3 Metodologia

Per a poder desenvolupar els objectius esmentats en l'apartat anterior, s'ha utilitzat com a document eix del treball el llibre (2). Aquest llibre utilitza el llenguatge *Python* i la llibreria *Keras* inclosa en el paquet de *TensorFlow* per crear les xarxes.

D'altra banda, s'ha de tenir en compte que l'entrenament d'una xarxa neuronal és un procés computacional costós, ja que les xarxes han de processar una immensa quantitat de dades per poder ser entrenades, de manera que els temps d'entrenament acostumen a ser bastant llargs.

Una de les estratègies que s'usa per reduir els temps d'entrenament és l'ús d'una targeta gràfica dedicada, ja que aquesta disposa de diversos nuclis de processament que permeten fer els càlculs tensorials d'una forma més eficient. Les característiques d'aquestes targetes pel que fa als recursos de memòria i velocitat de processament, així com la compatibilitat amb les llibreries *Keras* del paquet de *TensoFlow*, determinarà la seva idoneïtat. S'ha descartat la utilització de serveis en línia de targetes gràfica GPU, tal com *Google Cloud*, etc. pel cost econòmic que comporten.

Capítol 2

Xarxes neuronals

2.1 Definició

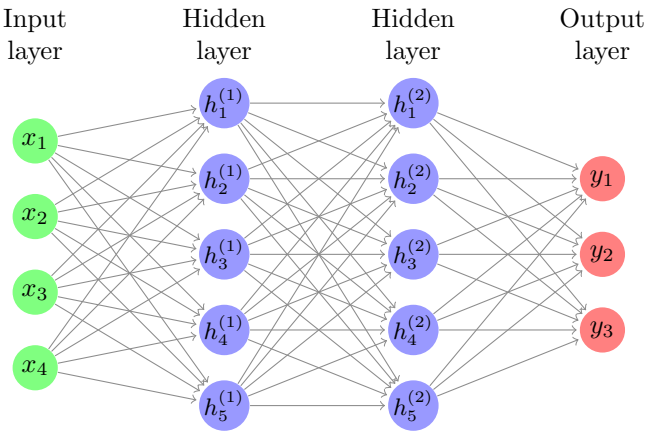
Una de les primeres definicions que es va donar de xarxa neuronal va ser la de Dr. Simon Harkin (1994, *Neural Networks: A Comprehensive Foundation* (3), NY Mc Millar, Pàg. 24):

> "Una xarxa neuronal és un procés distribuït massivament en paral·lel que té una tendència natural per emmagatzemar coneixement empíric i habilitar-lo per al seu ús..."

En essència, una xarxa neuronal és una funció que, mitjançant dades empíriques, és capaç de reconfigurar els seus paràmetres per tal d'aprendre de manera totalment autònoma els patrons que contenen aquestes dades.

Habitualment les xarxes neuronals consten de tres capes: la capa d'entrada, *Input Layer*; les capes ocultes, *Hidden Layers* i la capa de sortida, *Output Layer*. Les capes són conjunts de neurones, normalment representades verticalment. Les xarxes que tenen totes les neurones conectades amb les de la capa anterior i les de la capa següent reben el nom de xarxes denses.

Figura 2.1: Esquema d'una xarxa neuronal

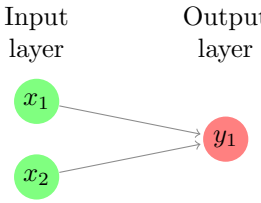


El nombre de neurones que té la capa d'entrada és igual a la magnitud de característiques del *Input* que permetran a la xarxa reconèixent-ne els diferents patrons. Tant el nombre de capes ocultes com la quantitat de neurones que tingui cada capa són paràmetres que variaran tant el rendiment de la xarxa com el cost d'entrenament. Per últim, la quantitat de neurones de la capa de sortida depèn del problema que es vulgui implementar. Per exemple, en un problema de classifcació hi haurà tantes neurones com classes.

2.2 Funcionament

Per explicar el seu funcionament, construirem l'exemple d'una xarxa neuronal trivial:

Considerem dues neurones x_1 i x_2 que poden prendre els valors 0 i 1. Aquestes dues neurones, situades en la capa d'entrada, estan connectades a una neurona de sortida que també prendrà valors 0 o 1 en funció de les primeres.



Diem que prendrà el valor 1 si i només si $x_1 = 1$ i $x_2 = 1$, en altres casos, el seu valor serà 0. Així doncs, els valors de la neurona de sortida y_1 seguiran aquesta taula:

		x_2	
		0	1
x_1	0	0	0
	1	0	1

Aquesta neurona funciona exactament igual que una porta lògica AND. Si connectem diverses neurones com aquesta entre elles podem crear una xarxa com la que hem vist al principi que ens permetrà resoldre problemes més complexos. La paraula *"Deep"* en *Deep Learning* fa referència a les múltiples capes de neurones que hi ha entremig de les capes d'entrada i les de sortida, les quals reben el nom de capes ocultes o *hidden layers*. És necessari remarcar que les xarxes neuronals que tractarem són xarxes densament connectades. Això vol dir que cada neurona està connectada amb totes les neurones de la capa anterior i que el seu valor influencia a totes les de la capa següent.

Es podria donar el cas que no totes les neurones d'una capa influissin de la mateixa manera sobre una neurona concreta del *layer* següent; per això la importància de cada connexió de neurones està ponderada per un pes o *weight*, que és un nombre racional llur magnitud quantifica aquesta importància. Introduint la notació si $w_{j,i}^{(l)}$ el primer subíndex fa referència a la neurona a la qual arriba, el segon és la neurona de la qual surt i el superíndex el *layer* al qual pertany.

Tornant a l'exemple anterior, si el valor de x_2 no tingués cap conseqüència sobre el valor de y_1, el pes $w_{1,2}^{(1)}$, que quantifica la influència del valor de x_2 sobre la neurona y_1, seria zero.

Siguin $x_1, ..., x_n \in [0, 1]$ els valors de les neurones de l'input layer i $w_{1,1}^{(1)}, w_{1,2}^{(1)}, ..., w_{1,n}^{(1)}$ els seus corresponents pesos cap a la primera neurona del layer següent, una primera aproximació al valor d'aquesta neurona seria:

$$\sum_i x_i w_{1,i}^{(1)} \,. \tag{2.1}$$

En moltes ocasions, diversos autors anomenen "activació"al valor que pren la neurona. Utilitzant aquest mateix llenguatge, s'acostuma a dir que les neurones que tenen una activació superior a un cert llindar estan activades.

Un altre paràmetre molt important que determina l'activació d'una neurona és el biaix. Aquest no és res més que un nombre $b_j^{(l)}$ que, sumant a l'expressió anterior, quantifica la facilitat amb la qual una neurona s'activa. Així doncs l'expressió (2.1) queda com:

$$\sum_i x_i w_{1,i}^{(1)} + b_j^{(1)} \,. \tag{2.2}$$

El tipus de neurones que hem utilitzat en aquest exemple s'anomenen perceptrons. Com hem vist, la seva característica principal és que només poden prendre dos valors: el 0 i l'1. Això presenta un problema. Tal com veurem més endavant, quan la xarxa neuronal està aprenent, canvien els valors de les neurones. Per intentar afinar aquest entrenament, interessa que els canvis en la xarxa siguin com més petits millor. Usant una xarxa amb perceptrons no es poden aconseguir canvis petits, ja que, com que és binària, un canvi de valor en la neurona implica un canvi radical en el funcionament de la xarxa. Per tant, per tal de poder fer canvis ínfims en la xarxa, s'opta perquè les neurones no siguin binàries sinó que prenguin qualsevol valor racional entre el 0 i l'1. Aquestes neurones s'anomenen neurones sigmoides.

Partint de l'equació (2.2), podem pensar que entrenar una xarxa neuronal és equivalent a trobar els valors dels pesos i dels biaixos corresponents a cada neurona que s'ajusten millor a les dades d'entrenament. Tanmateix, podem veure l'equació (2.2) com quelcom similar a una regressió lineal múltiple: Si partim de l'equació general d'una regressió lineal múltiple

$$Y_i = \beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + ... + \beta_n X_{in} \,, \tag{2.3}$$

observem que té exactament la mateixa forma.

Resumint, una xarxa neuronal entrenada correctament és capaç de predir uns *Outputs*, a partir d'uns certs *Inputs*. En el cas particular dels problemes de classificació, com el que hem vist anteriorment, el valor predit correspon a la probabilitat que té l'*Input* a pertànyer a una certa classe. La classificació final es realitza assignant al *Input* a la classe on té més probabilitat de pertànyer si la probabilitat és superior a un cert llindar.

2.2.1 Funció d'activació

La configuració de la xarxa neuronal que hem vist anteriorment presenta un problema: si encadenem diverses regressions lineals, acabem obtenint una regressió lineal. És a dir, podríem configurar una xarxa amb una sola neurona tal que produeixi els mateixos resultats. Això és degut al fet que la composició de funcions lineals és lineal:

Siguin f i g dues funcions lineals:

$$(g \circ f)(u + v) = g(f(u) + f(v)) = (g \circ f)(u) + (g \circ f)(v)$$

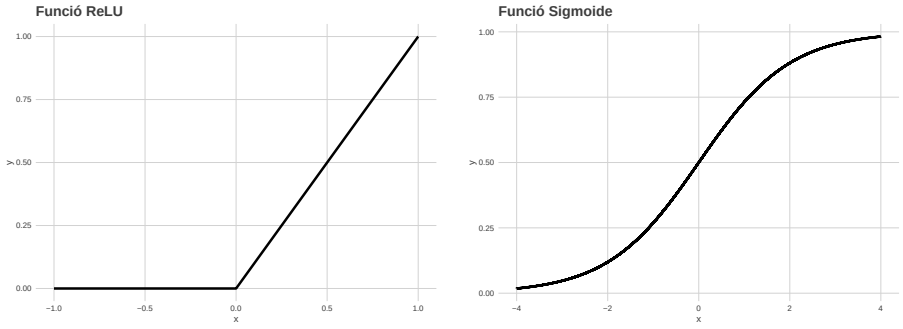
i

$$(g \circ f)(\lambda u) = \lambda g(f(u)) = \lambda (g \circ f)(u)$$

per les propietats de les funcions lineals.

Per tant, per tal de poder evitar aquest problema, cal que cada neurona introdueixi un últim càlcul més i afegeixi en l'expressió (2.2) una funció d'activació no lineal. Hi ha diversos tipus de funcions d'activació no lineals, entre les quals destaquen: la ReLU (*Rectified Linear Unit*) i la funció Sigmoide.

Figura 2.2: Gràfiques de les funcions d'activació ReLU i Sigmoide.



On la funció de la ReLU és:

$$f(x) = \begin{cases} x & \text{si x} \in [0, +\infty) \\ 0 & \text{si x} \in (-\infty, 0) \end{cases} . \tag{2.4}$$

I la de la sigmoide és:

$$f(x) = \frac{1}{1 + e^{-x}} . \tag{2.5}$$

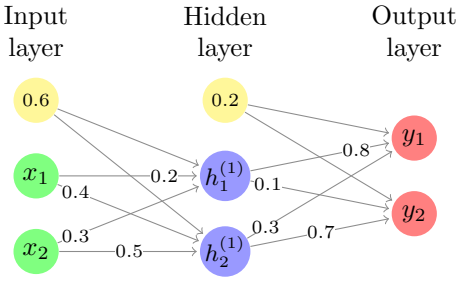
Finalment el valor de cada neurona serà calculat aplicant la funció d'activació $f(x)$ al valor obtingut per la suma ponderada (2.2).

Si ara generalitzem l'expressió, ens queda la següent equació:

$$x_j^l = f\left(\sum_{k=0}^{n_{l-1}} w_{jk}^{(l)} x_k^{(l-1)} + b_j^{(l)}\right) , \tag{2.6}$$

on n_{l-1} és el nombre de neurones en la capa $l-1$.

Exemplifiquem-ho amb la següent xarxa neuronal:



Si utilitzem els valors $\{x_1, x_2\} = \{0.45, 0.70\}$ com a valors d'entrada, denotem els valors dels pesos com el valor que hi ha associat a cada connexió i els valors dels biaixos representats amb cercles grocs. Calculem quin és el valor de la primera neurona del primer *hidden layer*:

Denotem com $z_i^{(l)}$ el valor de la suma ponderada descrita en l'equació (2.2).

$$z_1^{(1)} = x_1 w_{11}^{(1)} + x_2 w_{12}^{(1)} + b^{(1)} = 0.45 \times 0.2 + 0.7 \times 0.3 + 0.6 = 0.9 .$$

Utilitzant la funció d'activació sigmoide (2.5), tenim que:

$$h_1^{(1)} = \frac{1}{1 + e^{-z_1^{(1)}}} = \frac{1}{1 + e^{-0.9}} = 0.71095 .$$

Analogament trobem que $h_2^{(1)} = 0.75584$.

Per calcular els valors de les neurones del *layer* següent es fa servir exactament el mateix procediment:

$$z_1^{(2)} = h_1^{(1)} w_{11}^{(2)} + h_2^{(1)} w_{12}^{(2)} + b^{(2)} = 0.71095 \times 0.8 + 0.75584 \times 0.3 + 0.2 = 0.99551$$

$$y_1 = \frac{1}{1 + e^{-z_1^{(2)}}} = 0.73017\,.$$

De la mateixa forma, $y_2 = 0.69\,.$

2.2.2 Funció de Pèrdua

Un cop la xarxa calcula el valor de cada neurona de l'*Output Layer*, es mesura la precisió dels resultats amb funcions de pèrdua. Hi ha diversos tipus de funcions de pèrdua, però la més comuna és l'error quadràtic mitjà.

Siguin y_i els valors reals associats a cada x_i en el conjunt d'entrenament (és a dir els valors que s'han de predir) i $\hat{y}_i$ els valors predits mitjançant la xarxa neuronal, denotem l'error quadràtic mitjà (*MSE*) com:

$$MSE(y_i, \hat{y}_i) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2\,. \tag{2.7}$$

2.2.3 Entrenament

L'objectiu de l'entrenament és reduir al màxim l'error calculat amb la funció de pèrdua sense fer *Overfitting*, concepte que comentarem més endavant. Per tal d'aconseguir-ho, es troben els valors dels pesos i dels biaixos que el minimitzen. Per alleugerar la notació, anomenarem a aquests valors com els paràmetres entrenables. A l'inici de l'entrenament, els paràmetres entrenables són completament aleatoris, per tant, l'error obtingut és també arbitrari. L'entrenament consisteix en la modificació iterativa del valor dels paràmetres de forma que, progressivament, hi hagi un decrement de l'error.

Si tenim n paràmetres, podem formar un espai vectorial a $\mathbb{R}^n$ amb els valors que aquests poden prendre. Si afegim una dimensió més, l'error serà una funció $E : \mathbb{R}^n \to \mathbb{R}$ i per tant es representarà en una superfície de $\mathbb{R}^{n+1}$. L'objectiu de l'entrenament és trobar el mínim absolut d'aquesta superfície.

2.2.4 Gradient Descent

Sigui w un pes qualsevol, minimitzar l'error C respecte a aquest pes és equivalent a veure com s'ha de modificar w per tal que C sigui el mínim possible. Si expresséssim els valors que pren l'error en funció d'aquest pes, havent fixat prèviament els altres, obtindríem una funció de variable real. Minimitzar l'error seria equivalent a trobar el mínim global de la funció $C(w)$.

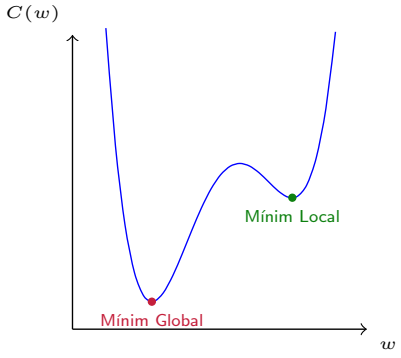


Figura 2.3: Representació de la funció de pèrdua respecte un pes w.

Extrapolant aquest exemple a una dimensió de paràmetres entrenables arbitrària, veiem que trobar el mínim de la funció $C(w)$ és equivalent a trobar la coordenada de l'hiperplà que formen els paràmetres entrenables que minimitza $C(w)$, on w ara denota el vector format amb tots els paràmetres entrenables.

Per tal de poder trobar el mínim absolut de la funció $C(w)$, ens interessa poder desplaçar els valors dels paràmetres entrenables cap a la direcció de l'hiperplà que fa que $C(w)$ descendeixi més. Per tant és essencial el concepte de gradient.

El gradient d'una funció és un camp vectorial on per cada punt de la superfície índica la direcció del pendent màxim. La definició formal és la següent: Sigui $f : \mathbb{R}^n \to \mathbb{R}$ definim el gradient $\nabla f : \mathbb{R}^n \to \mathbb{R}^n$ com

$$\nabla f(s) = \left(\frac{\partial f(s)}{\partial x_1}, \frac{\partial f(s)}{\partial x_2}, ..., \frac{\partial f(s)}{\partial x_n} \right) .$$

Per tant per poder minimitzar la funció de pèrdua, cal desplaçar els valors dels paràmetres cap a la direcció oposada del gradient. Si anomenem a x_n la posició en la iteració n-èssima, tenim:

$$x_{n+1} = x_n - s\nabla C(x_n) .$$

El paràmetre s s'anomena *step* o *learning rate* i és un coeficient que determina com serà de gran el pas que farà.

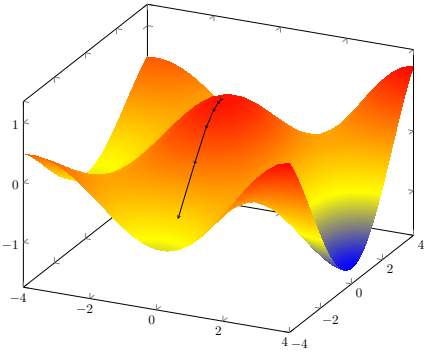


Figura 2.4: Representació del Gradient Descent amb dos paràmetres entrenables.

L'elecció del *step* no és pas trivial, ja que un *step* molt petit provoca una disminució en l'eficiència de l'entrenament i permet que el mètode pugui convergir cap a un mínim local. D'altra banda, un

step massa gran pot fer que l'entrenament no convergeixi mai cap a cap mínim, ja que pot ignorar la concavitat de la superfície que l'envolta.

2.2.5 Backpropagation

Per tal de poder calcular les derivades parcials de tots els paràmetres entrenables s'utilitza un algoritme anomenat *backpropagation*. Aquest consisteix a aplicar la regla de la cadena a tots els paràmetres de cada layer per tal de trobar-ne la seva derivada parcial. Ho exemplifiquem de la següent manera:

Considerem una xarxa neuronal amb L capes ocultes. D'acord amb el que hem vist anteriorment sabem que el valor que prendrà la neurona j-èssima de l'última capa serà el següent:

$$x_j^{(L)} = \sigma\left(\sum_{k=0}^{n_{L-1}} w_{jk} x_k^{L-1} + b^{(L)}\right).$$

Si considerem una funció de pèrdua $C(w,b)$, que anomenarem C per simplificar la notació, llavors tenim que les derivades parcials respecte un pes d'una neurona k de la capa $(L-1)$ són:

$$\frac{\partial C}{\partial w_{jk}^{(L)}} = \frac{\partial C}{\partial x_j^{(L)}} \frac{\partial x_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial z_j^{(L)}}{\partial w_{jk}^{(L)}}, \tag{2.8}$$

on hem usat la regla de la cadena i que les derivades dels altres pesos respecte el pes $w_{jk}^{(L)}$ són zero.

Anàlogament, amb els biaixos tenim:

$$\frac{\partial C}{\partial b^{(L)}} = \frac{\partial C}{\partial x_j^{(L)}} \frac{\partial x_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial z_j^{(L)}}{\partial b^{(L)}}. \tag{2.9}$$

Si calculéssim la derivada parcial de la funció cost respecte la $x_k^{(L-1)}$ veuriem que és un procés no trivial, ja que els canvis en aquest valor impliquen canvis en la resta de les neurones, per tant hauríem de sumar tots aquets canvis:

$$\frac{\partial C}{\partial x_k^{(L-1)}} = \sum_{j=0}^{n_{L-1}} \frac{\partial C}{\partial x_j^{(L)}} \frac{\partial x_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial z_j^{(L)}}{\partial x_k^{(L-1)}}. \tag{2.10}$$

Per calcular les derivades parcials en capes més profundes procediríem de manera similar, utilitzant la regla de la cadena.

2.3 Aspectes tècnics de les xarxes neuronals

En aquest apartat es tractaran diversos conceptes més orientats al funcionament pràctic de les xarxes neuronals. Es tractaran també alguns dels problemes que es poden presentar, així com algunes propostes per solucionar-los.

Conjunt d'entrenament

Per començar, tal com hem esmentat anteriorment, necessitem entrenar la xarxa per trobar els valors ideals dels paràmetres entrenables que s'adaptaran millor a les dades. La xarxa s'entrena sobre un conjunt de dades que s'anomena conjunt d'entrenament. Aquest acostuma a ser entre el 70 i el 80% de les dades totals. La resta de les dades totals s'utilitzen en el conjunt de test, tal com veurem posteriorment.

L'entrenament de la xarxa es realitza iterativament sobre tot el conjunt d'entrenament. Cada iteració que la xarxa fa sobre aquest conjunt s'anomena *epoch*. Tanmateix, ja que les dimensions del conjunt d'entrenament acostumen a ser desmesuradament grans, aquest s'acostuma a dividir en diferents parts anomenades *batches*. La versió del *Gradient Descent* adaptada a l'entrenament per *batches* s'anomena *Mini-batch Gradient Descent*.

Underfitting

Hi ha diversos problemes que podem tenir a l'hora d'entrenar una xarxa. El primer d'ells és l'*Underfitting*, que es dona quan els paràmetres de la xarxa no s'ajusten prou a les dades del conjunt d'entrenament. En casos com aquest, s'ha d'augmentar la complexitat de la xarxa i fer entrenaments més prolongats.

Overfitting

El cas contrari, molt més complex de resoldre, s'anomena *Overfitting* i té lloc quan els paràmetres entrenables s'ajusten excessivament a les dades del conjunt d'entrenament, fent impossible que el model pugui generalitzar i ser útil per a dades noves. Aquest problema no és exclusiu del *Deep Learning* sinó que és un problema general que poden tenir els models de *Machine Learning*. El seu tractament no és pas trivial i està subjecte a la naturalesa del model.

La solució més eficaç és augmentar el conjunt de dades d'entrenament. No obstant això, no sempre existeix aquesta possibilitat i s'ha de recórrer a altres tècniques. Una d'elles és el *Data Generator*, que, partint del principi anterior, consisteix a generar diverses imatges fent transformacions a les ja existents. Aquestes transformacions poden ser de diversos tipus, des de transformacions lineals fins a transformacions que variïn el color de les imatges. És important saber la idiosincràsia del problema que estem tractant a l'hora d'implementar un *Data Generator*, ja que en alguns casos ens pot perjudicar: un exemple trivial podria ser una xarxa que classifiqués si una persona està dreta o està tombada. Si amb l'ús d'aquesta tècnica es girés la imatge 90 graus, el funcionament de la xarxa es veuria afectat.

Conjunt de validació

Un dels problemes que genera l'*Overfitting* és que no és visible avaluant tan sols les dades d'entrenament. Durant l'entrenament de la xarxa, el més normal és que la precisió augmenti i els valors de la funció de pèrdua disminueixin. Encara que pugui semblar que el transcurs de l'entrenament estigui sent un èxit, no estem avaluant l'eficàcia del nostre model per manipular dades noves. Per solucionar aquest problema, el pas natural és crear un subconjunt de les dades d'entrenament que s'anomenarà conjunt de validació. D'aquesta manera podrem entrenar el model amb les dades d'entrenament restants i avaluar el resultat amb les dades de validació, que li són completament noves a la xarxa. Si veiem que hi ha un problema d'*Overfitting* o *Underfitting* es pot modificar la configuració de la xarxa, és a dir, la quantitat de capes, la quantitat de neurones que hi ha en cada una d'elles, etc. Aquests valors s'anomenen hiperparàmetres i modificant-los es pot augmentar o disminuir la complexitat de la xarxa i el grau en què s'adapta a les dades d'entrenament. Així doncs, l'objectiu és afinar els valors dels hiperparàmetres per tal de trobar els que aconsegueixen que la xarxa obtingui els millors resultats en el conjunt de validació.

Aquesta tècnica de partició de les dades s'anomena *Hold Out*, però no és l'única solució al problema. En cas que tinguéssim un conjunt de dades totals molt petit, es podria utilitzar (a canvi d'un cost computacional major) una tècnica anomenada *Cross Validation*, que aconsegueix els mateixos propòsits dividint totes les dades en *K-folds* i entrenant-ne $K-1$, deixant l'última *fold* per avaluar l'error. El procés es repetiria fins que cada *fold* s'hagués fet servir com a conjunt de validació.

Conjunt de test

Finalment, després que el model s'ajusti a les dades d'entrenament i hàgim afinat prou els hiperparàmetres perquè funcioni també sobre el conjunt de validació, podria semblar que és coherent

afirmar que el model funciona correctament i és capaç de generalitzar sobre dades noves. Tanmateix, això no té per què ser veritat.

Si recordem, l'entrenament d'una xarxa neuronal és l'afinació dels seus paràmetres entrenables usant les dades del conjunt d'entrenament. Així doncs, el procés que acabem de fer per afinar els hiperparàmetres no deixa de ser quelcom semblant al que fa la xarxa però manualment i, òbviament, amb moltíssims menys paràmetres. Per tant, comporta també el mateix problema que havíem comentat anteriorment: l'*Overfitting*. Com més afinem els hiperparàmetres per tal que el comportament sigui l'adequat en el conjunt de validació, més s'està adaptant la xarxa a la informació concreta d'aquest conjunt. Així doncs és necessari fer ús del conjunt de test, un subconjunt de les dades totals que només s'utilitzarà per fer una avaluació final del funcionament de la xarxa.

En resum, l'entrenament òptim d'una xarxa és un compromís entre l'optimització (és a dir, com de bé funciona el programa amb les dades d'entrenament) i la generalització, que és l'eficàcia amb què la xarxa funciona amb dades noves. Encara que en l'inici de l'entrenament, pel fet que el model encara té *Underfitting*, hi hagi una correlació entre aquestes dues característiques i augmentin paral·lelament, a mesura que l'entrenament avança, la xarxa es va adaptant progressivament a les dades d'entrenament i aquestes dues característiques deixen d'incrementar.

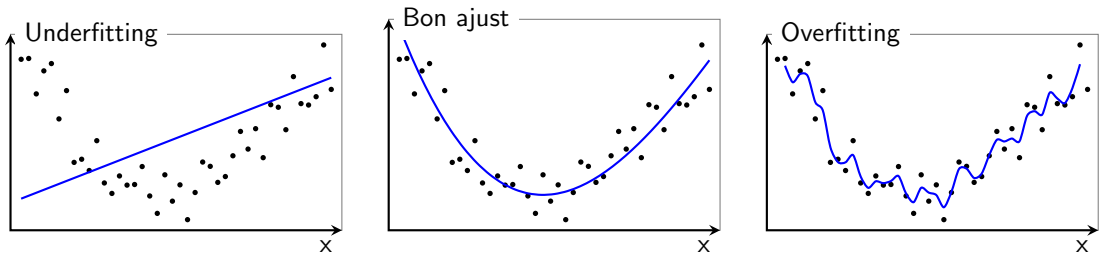


Figura 2.5: Representació de com serien l'Overfitting i l'Underfitting i, en contraposició, com seria un bon ajust.

L'optimitzador *Adam*

El *Gradient Descent* no és sinó un algoritme, en la seva versió més senzilla i primitiva, del conjunt d'algoritmes anomenats optimitzadors que s'encarreguen d'aproximar-se al mínim global de la funció de pèrdues.

L'*Adam* (*Adaptative Movement Estimation algorithm*) és un algorisme desenvolupat l'any 2014 per P. Kingma Diederik i Lei Ba Jimmy, autors de l'article *Adam: A method for stochastic optimization* (4). La particularitat essencial d'aquest algorisme és que el *learning rate* no és fixe sinó que va variant durant l'entrenament. Aquest optimitzador és descrit com una mescla entre dos dels algorismes més potents en aquella època: l'*AdaGrad* (Adaptative Gradient Algorithm) i el *RMSProp* (Root Mean Square Propagation). Ambdós algorismes utilitzen una estratègia que consisteix a designar un *learning rate* per cada paràmetre, fent que els valors de la funció de pèrdues decreixin molt ràpidament.

L'*Adam* parteix d'una estratègia utilitzada per l'*AdaGrad*: El *learning rate* està subjecte a les mitjanes del gradient de la funció de pèrdues, és a dir, en termes probabilístics, a l'estimació del primer moment del gradient. El tret distintiu que d'aquest nou algorisme és la incorporació del segon moment, és a dir, de la variància descentralitzada d'aquest mateix gradient.

A continuació mostrarem el funcionament de l'algorisme:

Per començar, introduirem la notació: cada iteració o *epoch* de la xarxa rebrà el nom de t, començant per $t = 0$. Els paràmetres a entrenar els denotarem com θ i la funció de pèrdues serà $f(\theta)$.

Així doncs, comencem definint la funció que ens mesurarà el gradient:

$$g_t = \nabla_\theta f_t(\theta_{t-1})\,.$$

A continuació, definim el primer moment, també usat pel *AdaGrad*, com m_t i amb valor inicial zero. Aquest valor està definit per un hiperparàmetre $\beta_1 \in [0,1)$ i la seva fórmula per $t > 0$ és:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t\,. \tag{2.11}$$

El paper del primer moment és essencial per tal de canviar el *learning rate* a cada pas: Aquest usa l'anomenat *Exponentially Weighted Moving Average* que fa que els valors de g_t per t més recents tinguin més pes en l'adaptació a les dades. Ho podem comprovar de la següent manera:

$$\begin{aligned}
&m_0 = 0\\
&m_1 = \beta_1 m_0 + (1-\beta_1) g_1 = (1-\beta_1) g_1\\
&m_2 = \beta_1[(1-\beta_1) g_1] + (1-\beta_1) g_2 = (1-\beta_1)[g_2 + \beta_1 g_1]\\
&m_3 = \beta_1(g_2 + \beta_1 g_1) + (1-\beta_1) g_3 = (1-\beta_1)(g_3 + \beta_1 g_2 + \beta_1^2 g_1)\\
&\quad\vdots\\
&m_n = (1-\beta_1)(g_n + \beta_1 g_{n-1} + \beta_1^2 g_{n-2} + \beta_1^3 g_{n-3} + ... + \beta_1^{n-1} g_1 + \beta^n g_0)\,,
\end{aligned}$$

per qualsevol $n \in \mathbb{N}$.

Observem que com que $\beta_1 \in [0,1)$, tenim que $\beta_1^n < \beta_1^{n-1}$ i que, per tant, se'ls hi dona més pes als valors més nous, on l'exponent de la β_1 és més petit.

El segon moment, que tal com hem dit abans és exclusiu d'aquest optimitzador, es defineix de forma semblant. S'anomena v_t i també estarà definit per un hiperparàmetre $\beta_2 \in [0,1)$. El seu valor és inicialment zero i per $t > 0$ pren els valors de la següent fórmula:

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2\,.$$

Com podem observar, la fórmula és molt similar a la del primer moment, per tant, els valors nous de v_t també donaran més pes als valors nous de g_t.

Com que tant m_t com v_t prenen el valor inicial zero, els primers passos de l'entrenament poden suposar un canvi molt brusc dels valors dels paràmetres, obrint la possibilitat a què la funció de pèrdues no disminueixi d'un cert valor. Per corregir aquest efecte s'implementen unes correccions del biaix al primer i segon moment:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \quad \text{i} \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t}\,.$$

L'últim pas consisteix en l'actualització els valors dels paràmetres usant tot el que hem calculat anteriorment:

$$\theta_t = \theta_{t-1} - \frac{\alpha * \hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon}\,.$$

On l'α s'anomena *stepsize* i té un valor recomanat de 0.001. Pel que fa als dos valors β_1, β_2 és necessari que prenguin valors propers a l'u, per tal que el seu funcionament sigui òptim. Les proves realitzades pels autors de l'article (4) conclouen que, generalment, els valors $\beta_1 = 0.9$ i $\beta_2 = 0.999$ funcionen correctament.

En aquest mateix article es parla de la divisió $\frac{\hat{m}_t}{\sqrt{\hat{v}_t}}$ com la relació senyal-soroll, també anomenada SNR (*Signal-to-noise ratio*). Quan aquest valor és molt petit, el pas que fa l'optimitzador és també molt petit. Això passa o bé quan hi ha molt soroll en les dades o bé quan estem molt a prop del mínim de la funció de pèrdues. Per tal que aquest valor estigui ben definit en el zero, s'hi suma una quantitat ε, que normalment té un valor de 10^{-8}.

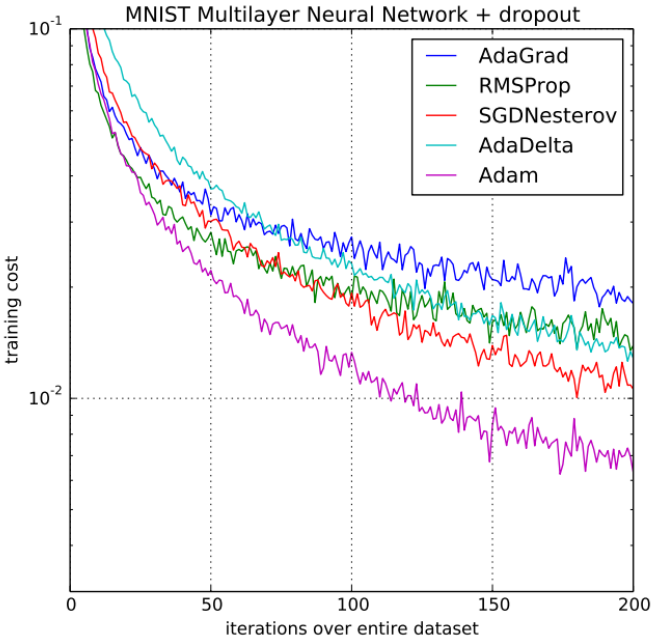


Figura 2.6: Comparació del rendiment de l'*Adam* amb els altres optimitzadors, segons l'article (4).

Capítol 3

Xarxes convolucionals

Les xarxes convolucionals són un tipus de xarxes neuronals molt usades en els problemes en què la informació ve disposada en forma matricial. Aquesta particularitat fa que sigui idònia en el tractament de les imatges.

En el *Deep Learning* les imatges són tractades com a tensors. Una imatge en blanc i negre és representada com una matriu on cada posició $[i, j]$ correspon al píxel de la imatge. S'utilitza una escala d'enters del 0 al 255 que indica l'escala de grisos, on el zero representa el negre i el 255 representa el blanc. Malgrat això, és habitual que en el *Deep Learning* els valors de cada píxel siguin representats en una escala del 0 a l'1. En el cas de les imatges RGB s'usa la mateixa estratègia però amb tres matrius diferents, una per a cada color. Cada color de la imatge s'expressa com una combinació lineal d'aquests tres.

3.1 Motivació

Tal com hem vist anteriorment, les xarxes denses aprenen sobre el conjunt de totes les dades. Per tant, l'ús de les xarxes denses per la visió per computació funciona de la següent manera: les xarxes troben els patrons que hi ha en les imatges trobant els diferents patrons amb què es disposen els píxels. Nogensmenys, això presenta un problema: l'efectivitat no és massa bona. Diverses imatges d'un mateix objecte poden tenir grans diferències pel que fa a la distribució dels seus píxels segons siguin les diferents condicions lumíniques amb les quals es va fer la fotografia, la resolució d'aquesta i altres factors que puguin intervenir. Per tant, neix la necessitat de poder distingir en una imatge els diferents elements i les diferents formes que hi apareixen, de forma anàloga a com ho farien els mecanismes de percepció visual del cervell.

Per donar resposta a aquestes necessitats es fa ús de les ja esmentades xarxes convolucionals. A efectes pràctics, les xarxes convolucionals aprenen, en primer lloc, a distingir els patrons més bàsics que tenen els objectes i, a continuació, la disposició que tenen aquests a l'hora de crear l'objecte.

Per exemple una xarxa que sigui capaç de reconèixer una cara aprendrà primerament a distingir les diferents parts que la conformen (és a dir: les celles, la boca, el nas...) i, posteriorment, trobarà quina disposició tenen aquestes parts i com, en conjunt, formen la cara. En contraposició, les xarxes denses aprenen quina és la disposició habitual dels píxels quan aquests representen una cara.

El gran avantatge que suposen les xarxes convolucionals és la invariabilitat respecte a les translacions de les imatges les xarxes són capaces de reconèixer els patrons ja apresos encara que es mostrin en una posició diferent de la qual s'han entrenat.

3.2 La funció de convolució

La base de les xarxes convolucionals és la convolució. Aquesta és una operació matemàtica on, a partir d'una imatge d'entrada i d'una matriu de nombres que es diu filtre o *kernel*, s'aconsegueix una imatge de sortida comunament anomenada "mapa de característiques". La idea és que cada configuració diferent dels valors que componen el filtre busca una característica determinada en la imatge. Així doncs, el mapa de característiques no deixa de ser un mapa on s'indica en quin lloc de la imatge original hi ha la característica concreta que busca el filtre.

Per definir-la formalment considerem f un filtre de dimensió $d \times d$ i I una imatge i definim la convolució com:

$$(f \otimes I)[i,j] = b + \sum_{u=-k}^{k} \sum_{v=-k}^{k} f[u,v] * I[i+u, j+v]\,, \tag{3.1}$$

on b representa el valor del biaix.

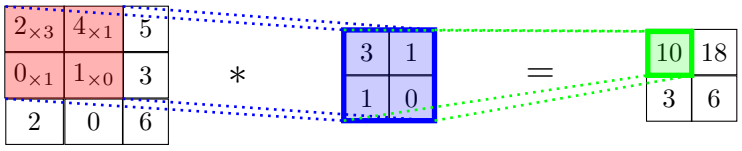
A continuació veurem un exemple del càlcul que realitza la convolució. Si considerem una imatge en blanc i negre els píxels de la qual, per fer més senzills els càlculs, prenen valors de 0 a 255:

2	4	5
0	1	3
2	0	6

I considerem també un filtre de dimensió 2×2 els seus valors del qual són:

3	1
1	0

La convolució resultant és:



Per tal de fer el càlcul de la convolució, hem de passar el filtre per diversos grups de píxels de la imatge. En el primer cas, passem el filtre pels quatre primers píxels de dalt a l'esquerra. Multipliquem el valor de cada píxel pel valor corresponent a la casella superposada del filtre i sumem els valors resultants. A continuació desplacem el filtre una posició cap a la dreta i realitzem el procés anterior. Quan hàgim acabat amb aquesta fila, realitzem el mateix procés per la següent fila fins a cobrir tota la imatge.

Els càlculs són els següents:

$$\begin{aligned} 2 \times 3 + 4 \times 1 + 0 \times 1 + 1 \times 0 &= 10 \\ 4 \times 3 + 5 \times 1 + 1 \times 1 + 0 &= 18 \\ 1 + 2 + 0 &= 3 \\ 3 + 3 &= 6\,. \end{aligned}$$

És important remarcar que, encara que en aquest cas hem aplicat un filtre de dimensió 2×2, les dimensions de filtres més comunes són: 3×3, 5×5 i 7×7.

Per entendre millor la convolució, considerem el següent exemple què li farem l'operació a una imatge de la torre de l'aigua de Sabadell per poder veure el mapa de característiques resultant.

Si considerem el següent filtre de dimensió 3×3:

-1	0	1
-1	0	1
-1	0	1

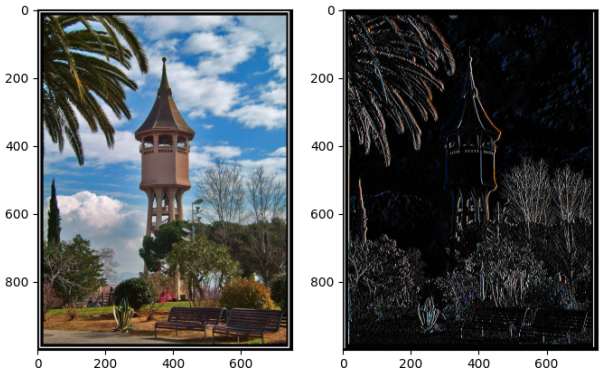
Aquest filtre és capaç de remarcar els contorns verticals dels diferents objectes de les imatges.

Quan el filtre passa per un conjunt de píxels que tenen tots el mateix valor i realitzem la convolució, tal com hem mostrat anteriorment, observem que el resultat de l'operació és zero. Això és degut al fet que els valors dels píxels que es multipliquen per la primera columna donen com a resultat uns valors negatius que anul·len els valors produïts per la multiplicació de la tercera columna del filtre pels valors de la imatge, que són els mateixos que els primers però en positiu. Per altra banda, com que la segona columna del filtre té valor zero, el seu producte amb els píxels de la imatge és zero i, per tant, no té cap efecte en la convolució.

En canvi, si ara suposem que el filtre està damunt d'un conjunt de píxels en què, per exemple, els píxels superposats per la primera columna del filtre tenen valor 0 i els píxels superposats per l'última columna del filtre tenen valor 255, situació que es correspon a un contorn vertical, el valor resultant de la convolució serà un nombre molt gran i, per tant, quedarà remarcat en el mapa de característiques. Seguint la notació que s'ha utilitzat en les xarxes neuronals, quan el resultat de la convolució és un nombre elevat i supera un cert llindar, es diu que aquella zona està activada en el mapa de característiques.

Si ho visualitzem:

Figura 3.1: A la esquerra la foto original i a la dreta un cop passat el filtre.



3.3 Funcionament de les xarxes convolucionals

Anteriorment, havíem vist que les xarxes neuronals denses entrenaven una sèrie de paràmetres per intentar trobar els valor d'aquests que minimitzava la funció de pèrdua. L'entrenament de les xarxes convolucionals funciona de manera similar, amb la particularitat que els valors dels filtres

de la xarxa són paràmetres entrenables, és a dir, que en cada iteració de l'entrenament canviaran els seus valors, donant un mapa de característiques diferent.

Entre els diferents hiperparàmetres que configuren les convolucions d'aquestes xarxes trobem la mida del filtre, la quantitat de filtres i el nombre de canals d'entrada. Aquest últim hiperparàmetre correspon al nombre de matrius a les quals se'ls hi passaran els filtres. Habitualment, les xarxes neuronals convolucionals estan compostes de diverses convolucions encadenades. En la primera d'aquestes el nombre de canals d'entrada és tres per les imatges en color i u per les imatges en blanc i negre, mentre que en les següents aquest valor és igual a la quantitat de filtres de la convolució anterior.

Un altre aspecte important a tenir en compte és que el mapa de característiques no té la mateixa dimensió que la imatge inicial, sinó que segueix la següent fórmula:

$$\text{Dimensió } \textit{Output} = \text{Dimensió } \textit{Input} - \text{Dimensió Filtre} + 1.$$

Per tal de modificar la dimensió del mapa de característiques existeixen diverses tècniques que veurem a continuació:

3.3.1 Padding

El *Padding* és una estratègia per obtenir un mapa de característiques de la mateixa dimensió que la imatge d'entrada. Aquest consisteix a afegir un nombre de files i de columnes amb zeros per tal de contrarrestar els efectes que provoca la convolució.

$$\begin{array}{|c|c|c|c|c|} \hline 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 2 & 4 & 5 & 0 \\ \hline 0 & 0 & 1 & 3 & 0 \\ \hline 0 & 2 & 0 & 6 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 \\ \hline \end{array} * \begin{array}{|c|c|c|} \hline 3 & 1 & 0 \\ \hline 4 & 0 & 1 \\ \hline 1 & 2 & 0 \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline 4 & 15 & 23 \\ \hline 7 & 15 & 33 \\ \hline 0 & 15 & 6 \\ \hline \end{array}$$

3.3.2 Strides

Els *Strides* són la quantitat de píxels que el filtre es desplaça cap a la dreta un cop acabada la convolució. En els exemples anteriors hem fet les convolucions amb *Stride*=1. Però, la quantitat de píxels que es pot desplaçar el filtre por ser major.

Com més gran sigui el *Stride*, més disminuirà la dimensió del mapa de característiques. Per tant si tenim en compte els efectes provocats pel *Padding* i pel *Stride*, la fórmula que ens dona la dimensió final del mapa de característica és la següent:

Siguin:

$$\begin{aligned} d_{in} &= \text{Dimensió Input} \\ d_{out} &= \text{Dimensió Output} \\ s &= \textit{Stride} \\ p &= \textit{Padding} \\ f &= \text{Mida del filtre} \end{aligned}$$

Llavors tenim que la dimensió del *Output* és:

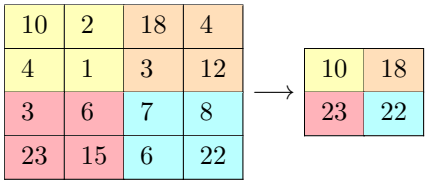
$$d_{out} = \frac{d_{in} + 2p - f}{s} + 1\,. \tag{3.2}$$

Fins ara les diferents tècniques que hem vist són modificacions que es fan a la convolució que havíem definit. No obstant això, existeixen tècniques alienes a aquesta que són capaces de reduir la dimensió del mapa de característiques. Una d'elles s'anomena *Maxpooling*.

3.3.3 Maxpooling

El *Maxpooling* és una tècnica que s'utilitza per disminuir dràsticament la dimensió del mapa de característiques. El seu funcionament és similar al de la convolució, en tant que consisteix en un filtre que va passant per conjunts de píxels de la dimensió d'aquest. En comptes de fer l'operació de convolució, el *Maxpooling* escull el valor màxim de cada grup de píxels compresos pel filtre. Aquest s'acostuma a fer amb *Stride*=2 i amb un filtre de dimensions 2×2, cosa que fa que la dimensió del mapa de característiques es redueixi a la meitat.

Si apliquem el maxpooling a l'exemple d'abans tenim:



Actualment, quan es tracta de reduir la dimensió del mapa de característiques s'acostuma a utilitzar més el *Maxpooling* que augmentar el *Stride* de la convolució.

Capítol 4

Super-Resolució

Des dels inicis del processament de les imatges, un dels problemes principals és el re-escalatge d'aquestes. Reduir la resolució de les imatges és un problema senzill, ja que no és res més que disminuir el conjunt total de les dades que conformen les imatges. En canvi, per augmentar-la, és necessari incrementar d'alguna forma la quantitat de píxels que té la imatge.

Existeixen diferents mètodes que ho aconsegueixen. Alguns d'ells no necessiten d'ús de tècniques d'intel·ligència artificial per tal de procedir.

El mètode més senzill que aconsegueix aquest propòsit és el *nearest neighbours* la traducció literal del qual és veïns propers. Aquest consisteix a augmentar la quantitat de píxels d'una imatge duplicant els píxels existents:

10	18
23	22

$\longrightarrow$

10	10	18	18
10	10	18	18
23	23	22	22
23	23	22	22

Aquesta tècnica no és ben bé el que s'esperaria d'un programa de super-resolució ja que, encara que augmenti la quantitat de píxels, no millora la qualitat de la imatge. Per tant, aqueta tècnica acaba per ser poc utilitzada i només s'usa en alguns contextos molt concrets en el qual es volen emfatitzar els píxels, com ara el Pixel Art. Un exemple utilitzant aquesta tècnica seria:

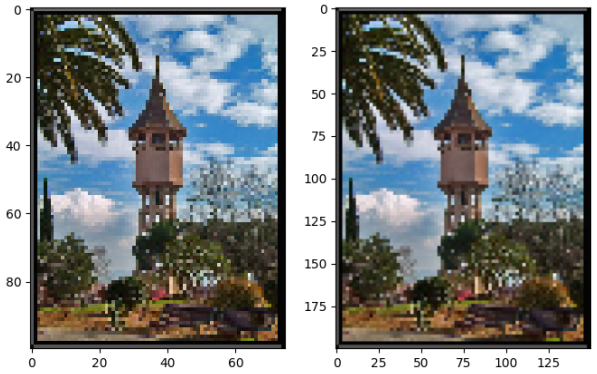


Figura 4.1: Super Resolution amb Nearest Neighbours.

Malgrat això, existeixen altres algoritmes que, sense utilitzar la intel·ligència artificial, aconsegueix-

xen millorar la qualitat de les imatges. Un d'aquests algoritmes s'anomena *bilinear interpolation*. Aquest aconsegueix crear uns píxels entremig dels ja existents que segueixen una funció d'interpolació bilinear de forma que els seus valors estan entremig dels valors dels píxels que els envolten.

Exemplificant-ho:

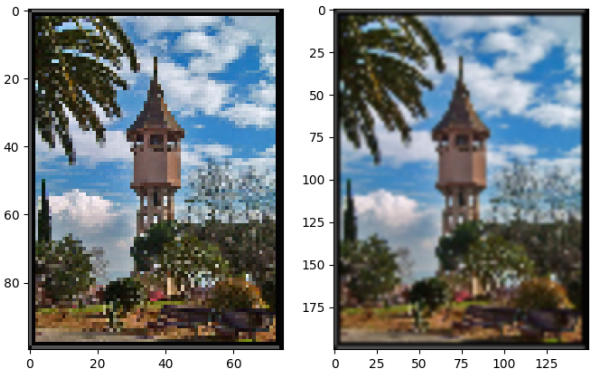


Figura 4.2: Super Resolution amb bilinear interpolation.

Per aconseguir uns resultats més interessants, cal fer ús de les xarxes neuronals convolucionals. Tal com hem comentat anteriorment, aquestes són capaces de detectar els diferents patrons de les imatges i afegir els píxels d'acord amb aquests patrons.

4.1 FSRCNN

4.1.1 Introducció

Per tal d'aconseguir que una xarxa convolucional resolgui el problema de la super-resolució, es poden implementar una gran varietat de dissenys. Com que entrenar una xarxa convolucional de dimensions considerables té un cost computacional molt elevat, la millor idea és començar amb un disseny que sigui eficaç però amb un cost relativament reduït. Un exemple és l'algorisme fet per Dong *et al* (1), el FSRCNN *(Fast Super-Resolution Convolutional Neural Network)*.

Per poder implementar aquest algorisme, primer cal introduir un nou concepte que és el de la convolució transposada. Aquesta és una operació que, a diferència de la convolució, augmenta la imatge de sortida.

Si considerem la següent imatge:

2	4
0	1

I considerem el següent nucli de convolució transposada:

3	1
1	5

El càlcul de la convolució transposada seria el següent:

$$\begin{bmatrix} 2 & 4 \\ 0 & 1 \end{bmatrix} * \begin{bmatrix} 3 & 1 \\ 1 & 5 \end{bmatrix} = \begin{bmatrix} 6 & 2 & \\ 2 & 10 & \\ & & \end{bmatrix} + \begin{bmatrix} & 12 & 4 \\ & 4 & 20 \\ & & \end{bmatrix} + \begin{bmatrix} & & \\ 0 & 0 & \\ 0 & 0 & \end{bmatrix} + \begin{bmatrix} & & \\ & 3 & 1 \\ & 1 & 5 \end{bmatrix} =$$

$$= \begin{array}{|c|c|c|} \hline 6 & 14 & 4 \\ \hline 2 & 17 & 21 \\ \hline 0 & 1 & 5 \\ \hline \end{array}$$

Com es pot observar, la imatge resultant de la convolució transposada té una mida més gran que la inicial.

Seguint la notació que hem usat a l'equació (3.2), la dimensió del output de la convolució transposada és:

$$d_{out} = (d_{in}s) + f - 2p\,. \tag{4.1}$$

Tal com hem dit anteriorment, l'objectiu de les FSRCNN és aconseguir resoldre el problema intentant minimitzar al màxim els recursos computacionals. Aquest algorisme es basa en el seu predecessor que és el SRCNN (*Super Resolution Convolutional Neural Network*), creat pel mateix equip de recerca l'any anterior (5).

Entre els diversos algorismes existents, el SRCNN suposa un increment en la velocitat de l'entrenament de la xarxa. El FSRCNN incrementa encara més la seva eficiència. Per aconseguir-ho, la xarxa reconeix els patrons bàsics de les imatges originals. En canvi, sobre l'algorisme anterior, es realitza primerament un preprocessament de la imatge que consisteix en la interpolació bicúbica, un augment de la resolució similar als que hem vist anteriorment, que fa ús de la interpolació cúbica en les dues dimensions de la imatge.

4.1.2 Estructura de la xarxa

Les estructures de les xarxes esmentades segueixen el següent esquema 4.3. Per fixar una notació, a la convolució li direm $Conv(f_i, n_i, c_i)$ i a la Convolució Transposada, també anomenada a vegades Deconvolució, li direm $DeConv(f_i, n_i, c_i)$ on f_i, n_i, c_i representen la mida del filtre, el nombre de filtres i el nombre de canals respectivament:

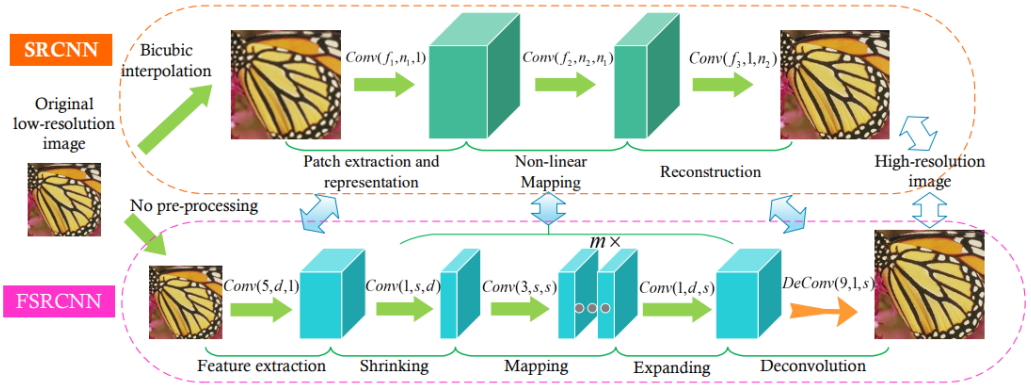


Figura 4.3: Estructura de les xarxes SRCNN i FSRCNN.

La idea general és crear una xarxa neuronal convolucional que sigui capaç d'augmentar la resolució d'una imatge d'unes mides d'entrada i sortida prefixades. Aquesta xarxa s'entrenarà amb un conjunt d'imatges que tindran una resolució igual a la resolució de la imatge de sortida. Es reduirà la mida d'aquestes imatges per tal que serveixin com a *Input* de la xarxa. Posteriorment, la xarxa aprendrà a augmentar-los la resolució i comprovarà l'error amb la funció de pèrdua i la imatge amb la mida original.

A continuació aprofunditzarem en les diferents parts que componen el FSRCNN.

Extracció de Característiques

En aquest apartat es realitza un mapa de característiques (*feature map*) amb la imatge original en baixa resolució. Tal com hem comentat, realitzar l'extracció de característiques sobre la imatge en baixa resolució és el punt clau que permet realitzar l'algorisme amb el mínim cost computacional. Això és degut al fet que la imatge d'entrada té una mida més petita i, per tant, el filtre haurà de passar per menys píxels.

S'utilitza un filtre de dimensió 5 i *stride* 1. Es contraresten els efectes de la disminució de la dimensió de la imatge usant el *Padding*, de manera que la dimensió final de l'*Output* és la mateixa que la inicial. Pel que fa al nombre de filtres, serà un paràmetre que anomenrem d i el nombre de canals de la convolució, només n'hi haurà un.

Shrinking

El pas natural després de fer l'extracció de característiques seria fer el *mapping*, però, la quantitat de canals (també anomenada com "profunditat") és massa gran, ja que hi ha tants canals com filtres hàgim utilitzat en la convolució anterior. Com que hem usat d filtres, tindrem d canals. Per reduir aquest nombre farem el *Shrinking*. Aquest pas consisteix a fer una convolució amb una quantitat de filtres s tal que $s << d$ que permetrà obtenir un mapa de característica llur profunditat sigui molt menor.

En aquesta combinació utilitzarem un sol filtre i amb stride 1. No cal usar *padding*, ja que la dimensió final és la mateixa que la inicial.

Mapping no lineal

Aquest apartat és el més important i és el que relaciona les característiques extretes en el primer apartat amb el resultat final de la convolució transposada. Per fer-ho es fan m convolucions amb s filtres de grandària 3 en el que se li introdueixen els mapes de característica amb profunditat s. En aquest pas la mida del filtre pot ser variable i podria ser augmentada, per exemple, a una dimensió de 5.

La quantitat de convolucions que es realitzen també pot ser variable. L'augment d'aquestes provocarà també un augment en el cost computacional de l'entrenament oferint, però, un increment en la complexitat de la xarxa. En l'article (1) es recomana el valor de $m = 4$.

Expansió

En aquest cas es fa el procés invers al *Shrinking*, és a dir que s'eixampla la profunditat del mapa de característiques. La quantitat de canals d'entrada seran s, ja que coincideix amb la quantitat de filtres de l'apartat anterior i la quantitat de filtres de dimensió 1 que utilitzarem serà d.

Aquest pas és necessari, ja que si generéssim directament la imatge des del mapa de característiques obtingut del *mapping*, hi hauria una disminució considerable en la qualitat de la super-resolució.

Convolució Transposada

Per acabar, queda usar la convolució transposada, que és l'element clau d'aquesta xarxa. Tal com hem vist anteriorment, aquesta és una operació que permet augmentar la dimensió del mapa de característiques. Com podem observar en l'equació (4.1), el *stride* de la convolució transposada multiplica la dimensió de l'*Input*, augmentant així d'una forma proporcional la dimensió de l'*Output*. Per tant, determinarem l'augment de la resolució que farà la xarxa a partir d'aquest *stride*.

Configuracions de la xarxa

La funció de pèrdua usada és la que recomana l'article, el *MSE*. Pel que fa a l'optimitzador, en comptes d'usar el *Stochastic Gradient Descent* (tal com recomana l'article), hem utilitzat l'*Adam*, que, com hem comprovat anteriorment, ens garantirà una convergència més ràpida. Per últim, hem

usat la ReLU com a funció d'activació, en comptes de la PReLU (una modificació de l'anterior), que es recomana en l'article.

4.1.3 Entrenament amb la llibreria MNIST

Hem debutat en la tècnica de la super-resolució amb la llibreria MNIST *(Modified National Institute of Standards and Technology)*. Aquesta consta de diverses imatges en blanc i negre de nombres dibuixats manualment i etiquetats amb el valor al qual representen. És un conjunt de dades molt ampli, ja que consta de 60000 imatges d'entrenament i 10000 de test.

Hem realitzat un entrenament de 500 *epochs* on els valors dels hiperparàmetres de la xarxa han estat:

$$\begin{aligned} d &= 56 \\ s &= 12 \\ m &= 4 \end{aligned}$$

L'*Input* han estat imatges de mida 14×14 i d'un sol canal, ja que són en blanc i negre. D'altra banda, l'*Output* han estat les imatges amb les dimensions habituals de la llibreria MNIST, és a dir, 28×28. Els resultats són els següents:

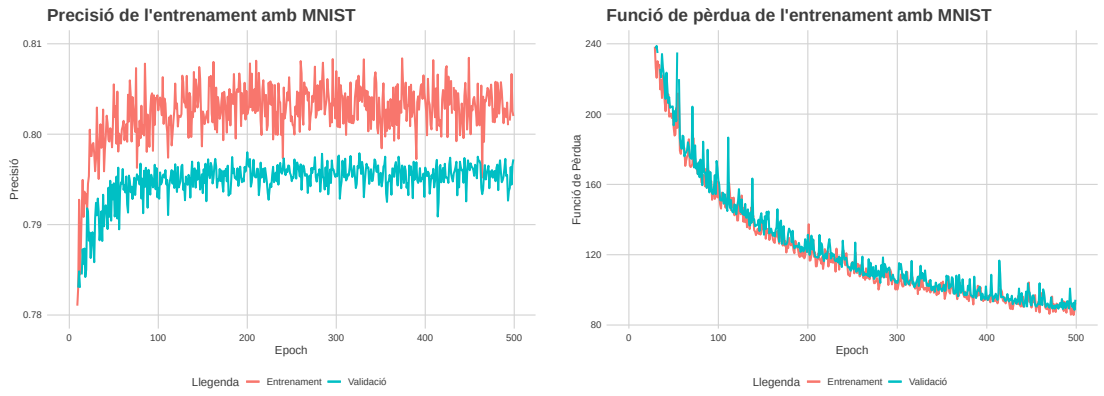


Figura 4.4: La variació de la precisió i de la funció de pèrdua en cada *epoch*.

Cal remarcar que els valors de la funció de pèrdues són molt elevats, ja que els píxels de les imatges prenen valors entre 0 i 255 en comptes de 0 i 1, com s'acostuma a veure. Això és degut al fet que els resultats són notablement més bons d'aquesta manera.

D'altra banda hem obtingut bons valors pel que fa a la precisió. Com és habitual, la funció de pèrdua és més alta i la precisió més baixa en el conjunt de validació, que és quan el nostre model està generalitzant.

Per comprovar el funcionament de la xarxa, hem dibuixat el nombre 3 amb un programa d'edició d'imatges, li hem reduït la mida i, posteriorment, la hi hem augmentat utilitzant la xarxa. Aquest és el resultat:

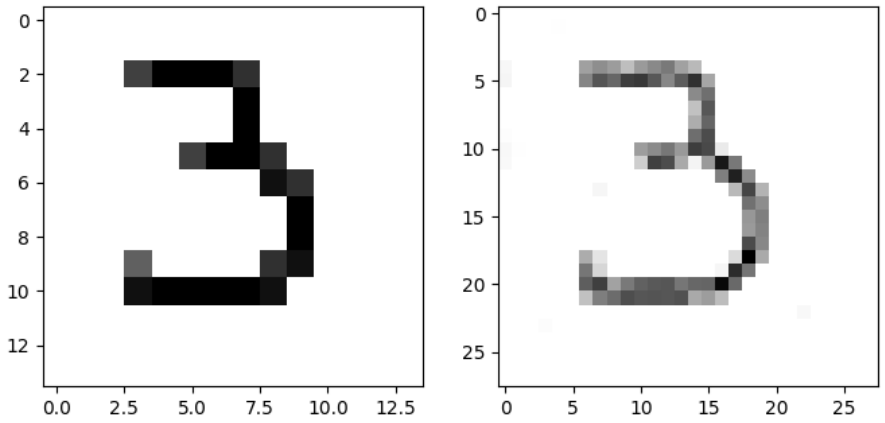


Figura 4.5: Resultat pràctic de la xarxa FSRCNN.

4.1.4 Entrenament amb la llibreria BSDS300

Veient els resultats obtinguts de les proves anteriors, podem comprovar que l'estructura de la xarxa funciona correctament amb imatges en baixa resolució i en blanc i negre. El següent pas seria repetir el mateix procediment, però aquest cop utilitzant un conjunt d'imatges en color i més grans.

Per tal d'aconseguir-ho, utilitzarem la llibreria BSDS300 (Berkeley Segmentation Data Set 300) que consta de 200 imatges d'entrenament i de 100 imatges de test.

Aquest cop, en l'entrenament continuarem utilitzant 500 *epochs* i els mateixos valors dels hiperparàmetres. Haurem de canviar, però, la forma de les dades d'input: En aquest cas estem introduint imatges de dimensió 150×150 que tenen 3 canals, ja que són imatges RGB. Pel que fa a l'*Output*, de moment continuarem ampliant la resolució el doble, per tant, les imatges que surten de la xarxa tenen una mida de 300×300.

Així doncs, obtenim els següents resultats:

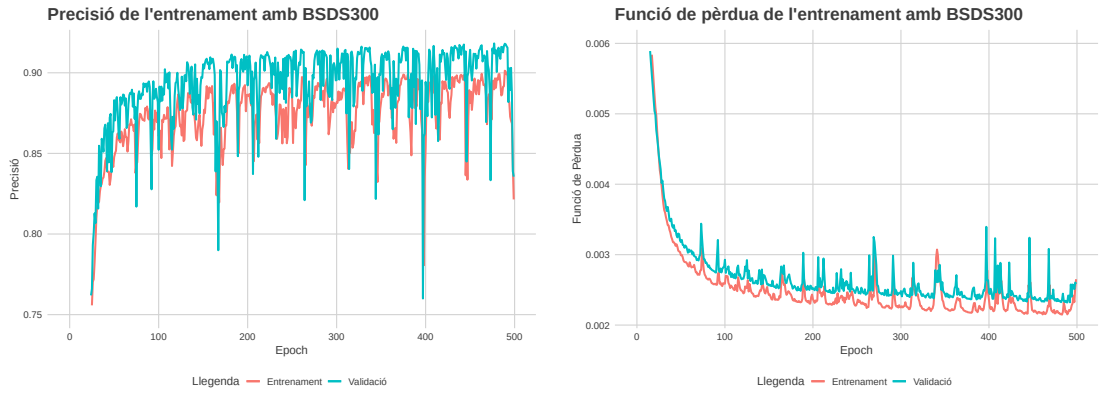


Figura 4.6: La variació de la precisió i de la funció de pèrdua en cada *epoch*.

Si provem el funcionament de la xarxa amb una foto del conjunt de test tenim:

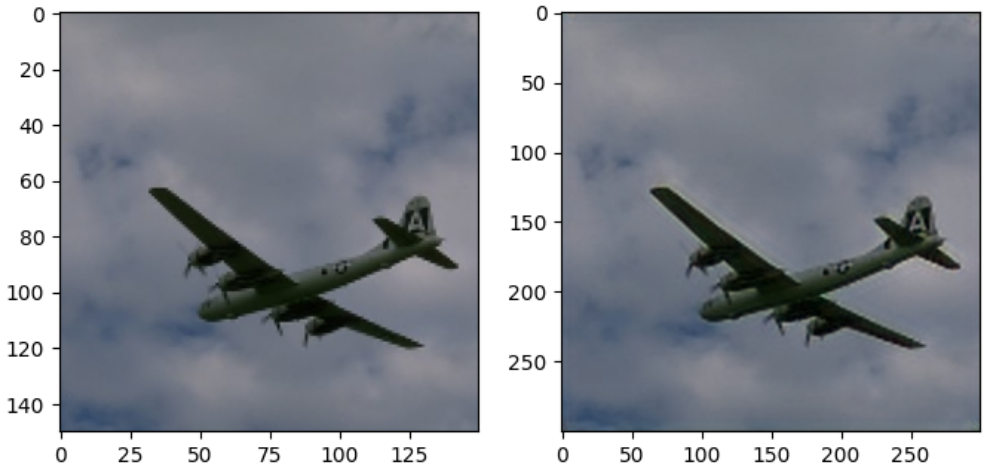


Figura 4.7: Mostra del resultat amb BSDS300.

Com podem observar els resultats no són massa bons. S'aprecien diverses distorsions en la imatge final que són prou notables en les línies rectes com ara en les ales de l'avió. És necessari evidenciar que s'està entrenant el model amb una quantitat ínfima d'imatges i que, per tant, és normal que el seu rendiment sigui bastant pobre.

Una millora que podríem implementar sense canviar el conjunt d'imatges seria fer un *Data Augmentation* que, tal com hem comentat anteriorment, consisteix en la generació de diverses imatges noves a partir de transformacions lineals d'imatges ja existents.

Un primer exemple és la foto de l'ocell que es veurà a continuació: la creació d'aquesta nova imatge i la seva incorporació en el conjunt d'entrenament fa que la xarxa sigui ara capaç de distingir i reconèixer un ocell, encara que estigui mirant cap a l'altra direcció.

Figura 4.8: Exemple del que fa el *Data Augmentation*, a l'esquerra l'original i a la dreta el modificat.

Data Augmentation

Si entrenem el model utilitzant el *Data Augmentation*, obtenim una millora en el rendiment: observem com els valors de la funció de pèrdua del conjunt de validació han disminuït.

Pel que fa a l'entrenament, hem seguit amb la mateixa configuració dels hiperparàmetres que hi havia anteriorment. No obstant, és important remarcar que el temps d'entrenament ha sigut molt més llarg.

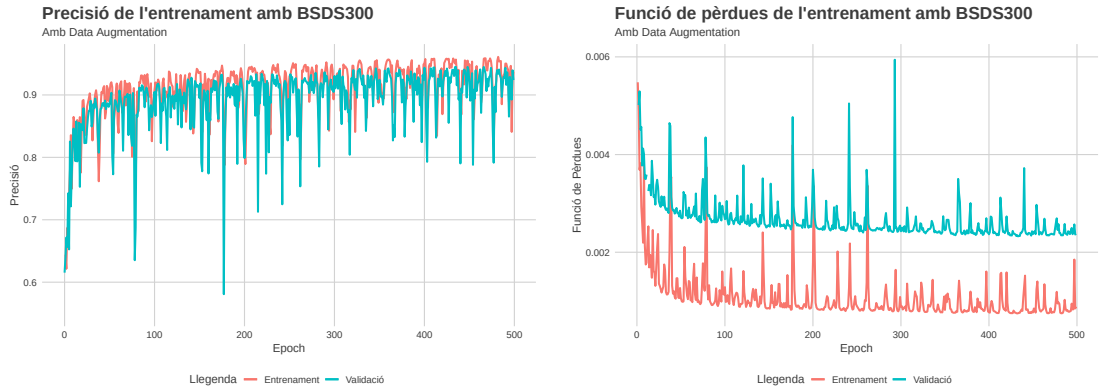


Figura 4.9: La variació de la precisió i de la funció de pèrdua en cada *epoch*.

Si traduïm aquests resultats a l'exemple anterior, observem una clara millora:

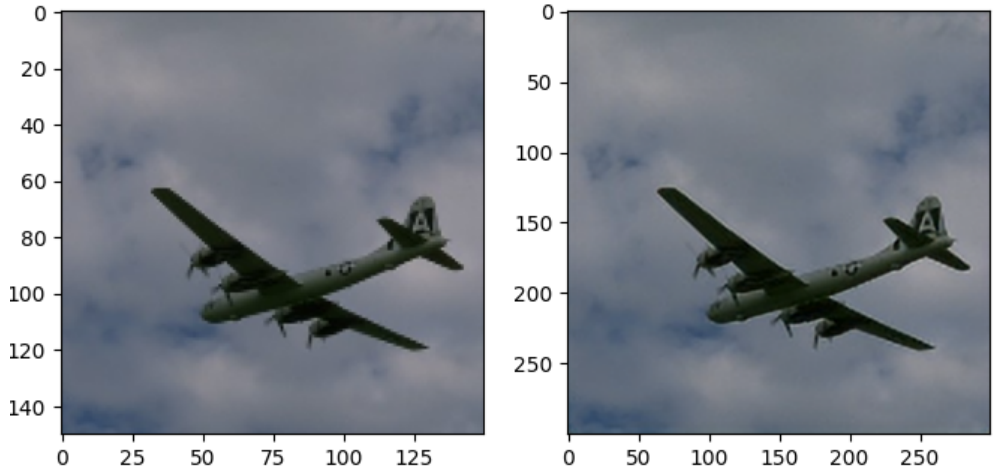


Figura 4.10: Augment de la resolució amb Data Augmentation.

Contemplem que, encara que continuï havent-hi distorsions en les imatges, aquestes s'han reduït notablement i, per tant, la imatge és més clara que en l'exemple anterior.

L'entrenament amb *Data Augmentation* és útil en situacions en què no és possible obtenir més dades. Com que nosaltres volem augmentar la resolució d'imatges genèriques, tenim una gran varietat de *datasets* molt més amplis que ens permetran fer una millor tasca.

4.1.5 Llibreria *Flickr*

La llibreria *Flickr* conté una gran varietat d'imatges diverses, concretament més de 31000. Malgrat això, hi ha dues diferències amb els *datasets* que hem usat anteriorment. Per començar, totes les imatges estan juntes, és a dir, que les haurem de dividir en els conjunts d'entrenament, validació i test. A més, el tret característic més important i que, potencialment, pot donar més problemes és l'ordenació de les dades. Si ens fixem hi ha moltes imatges similars seguides, cosa que pot ser un problema:

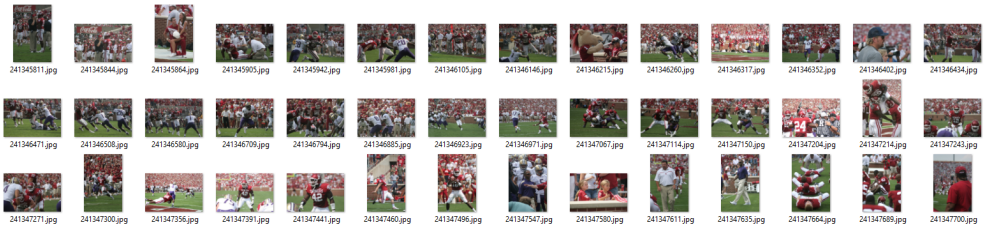


Figura 4.11: Mostra on s'observen moltes imatges seguides de futbol americà.

Si dividim el conjunt de dades totals en els tres subconjunts esmentats anteriorment, es donarà el cas de que totes les imatges d'una certa cosa hauran anat a parar a un d'aquests tres subconjunts. Això farà que, quan s'avaluin les dades, la naturalesa dels *datasets* sigui massa diferent i l'error que tindrà el model, quan s'avalui la seva capacitat de generalització amb els conjunts de validació i test, serà molt alt.

Una altra qüestió que s'ha de tenir en compte és que la xarxa aprèn millor quan les imatges sobre les que s'entrena són aleatòries. Això vol dir que un seguit d'imatges en les que apareix una mateixa cosa perjudica molt l'aprenentatge.

La solució al problema ha estat crear un vector que contingui aquestes imatges, escartejar-lo i assignar a cada carpeta de cada *dataset* la part proporcional dels elements del vector que els hi corresponen.

Com que el conjunt d'imatges és d'unes dimensions molt grans, no hi ha necessitat de fer un *Data Augmentation*. A més, un entrenament amb tants *epochs* com els que s'han vist anteriorment, trigaria més d'un dia a completar-se. Per tant, hem fet un entrenament de 50 *epochs* i els resultats són els següents:

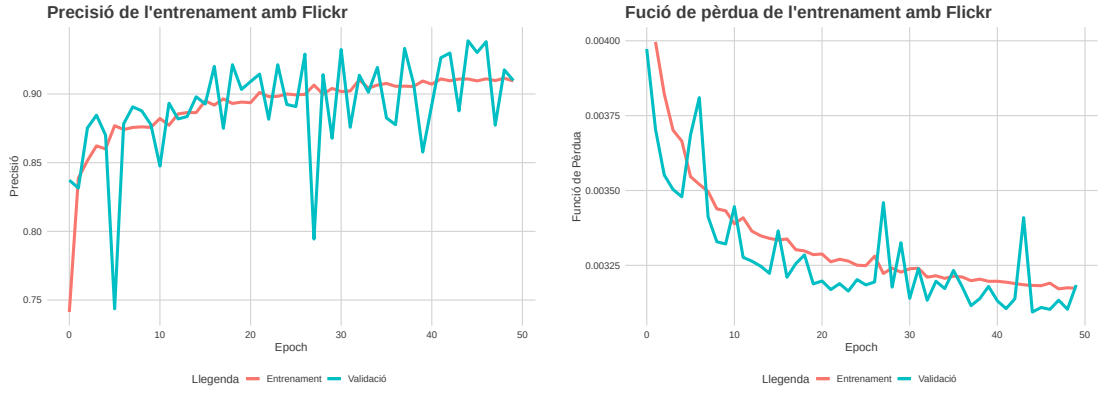


Figura 4.12: La variació de la precisió i de la funció de pèrdua en cada *epoch*.

Com podem observar, hi ha una millora respecte el conjunt d'imatges anterior. S'ha de remarcar, però, que és un gràfic bastant poc comú, ja que els resultats de la xarxa en el conjunt de validació semblen ser millors que els de l'entrenament. Tot i això, és important dir que l'escala d'aquests gràfics és molt petita, és a dir, que encara que sembli que en alguns llocs hi hagi molta diferència entre els resultats de l'entrenament i la validació, la xarxa funciona d'una forma molt similar en aquests dos conjunts. Per tant, podem afirmar que generalitza molt bé.

Podem posar en pràctica el funcionament de la xarxa usant el mateix exemple:

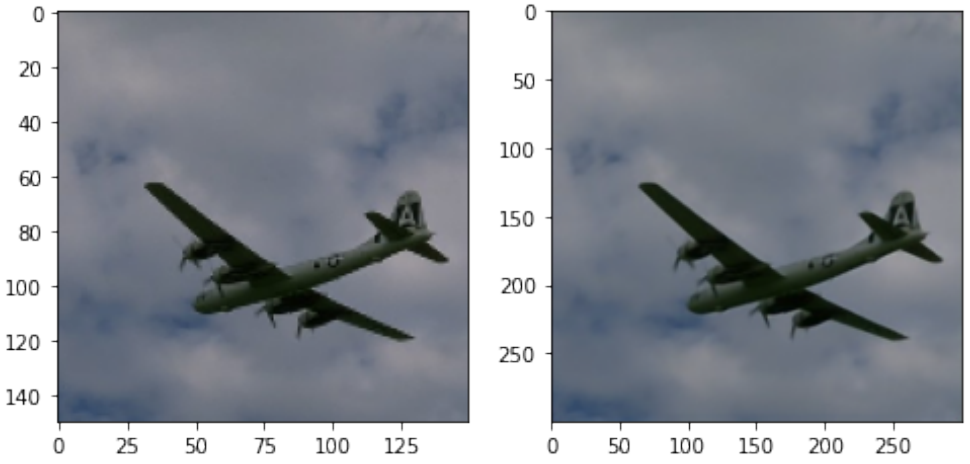


Figura 4.13: Visualització d'un exemple pràctic de la xarxa entrenada amb *Flickr*.

Quadruplicar la resolució

Aprofitant que disposem d'un conjunt de dades d'aquestes dimensions, seria interessant poder comprovar si és possible utilitzar la mateixa estructura de la xarxa per tal d'augmentar la resolució d'una imatge quatre vegades.

Per poder aconseguir aquests resultats, és interessant augmentar la complexitat de la xarxa. Una manera de fer-ho és augmentar el *Mapping* del FSRCNN. Si augmentem el valor de la m, que inicialment era 4, a 7, aconseguiem els següents resultats:

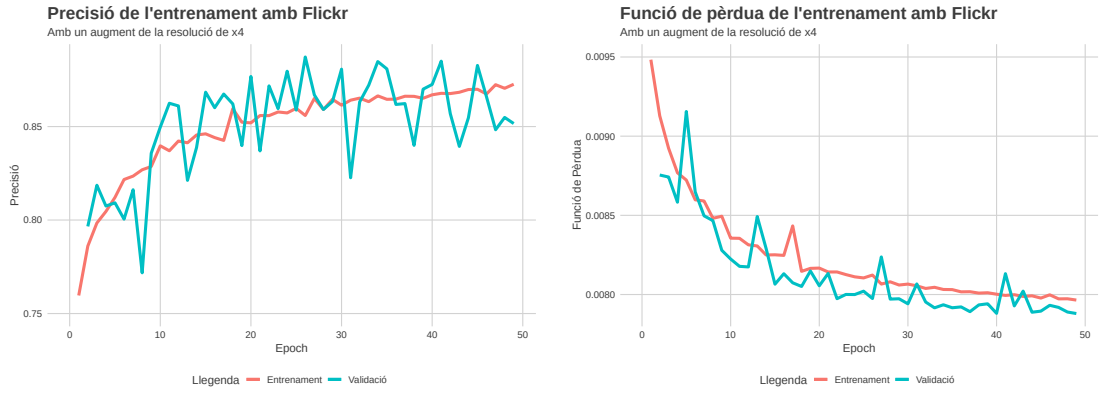


Figura 4.14: La variació de la precisió i de la funció de pèrdua en cada *epoch*.

Podem comprovar amb un exemple pràctic com la xarxa és capaç de quadruplicar la resolució de la imatge de l'ala d'una papallona que no ha vist mai:

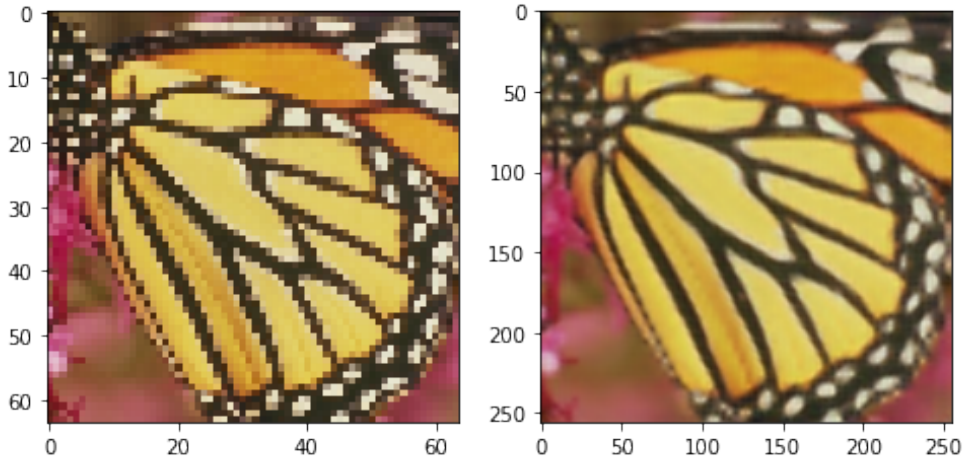


Figura 4.15: Augment de la resolució en una ala d'una papallona.

Capítol 5

Conclusions

Des dels inicis de la fotografia digital, augmentar la resolució d'una imatge, ha estat, fins fa relativament poc, una tasca impossible d'aconseguir de forma satisfactòria. Tal com hem comentat, els diferents mètodes que ho intenten sense fer ús del *Deep Learning* generen unes imatges, que si bé tenen la quantitat de píxels desitjada, són borroses i no s'hi acaben de distingir prou bé els seus elements.

El *Deep Learning* ha suposat un canvi de paradigma en la computació clàssica excel·lint completament en moltes de les tasques en les quals s'ha utilitzat. El seu bon funcionament dona resultats que haguessin semblat del tot impossibles fa només unes decades. El camp de la visió per computació està lluny de ser una excepció i és en aquest on trobem gran part dels resultats més espectaculars, un dels quals és la super-resolució. Aquesta, encara que realment ha sigut objecte de molt pocs estudis, ha sigut perfeccionada de tal forma que els seus avenços han estat útils, no només en els camps més artístics com ara la fotografia o en la renderització multimèdia com en els videojocs (NVIDIA DLSS), sinó en altres més tècnics com el diagnòstic clínic (6).

Pel que fa a les diverses xarxes que hem creat, és interessant veure l'evolució que han tingut els diferents dissenys de les xarxes, així com els diversos problemes que hi ha hagut en el seu funcionament. Inicialment, l'objectiu era dissenyar una xarxa convolucional capaç d'augmentar la resolució de les imatges de la llibreria MNIST. Aquest propòsit s'ha assolit de forma senzilla implementant la xarxa amb la mateixa configuració d'hiperparàmetres que s'esmentava en l'article (1). En canvi, en la generalització d'aquesta mateixa qüestió a imatges quotidianes usant la llibreria BSDS300, els resultats han estat considerablement millorables. S'ha pogut comprovar de forma pràctica com l'aplicació de diverses estratègies aconseguien millorar substancialment el resultat inicial. Al principi, s'ha provat de realitzar un *Data Augmentation* sobre el conjunt de BSDS300, i, si bé hi ha hagut millora, els resultats encara podrien haver estat més bons. L'estratègia que ha obtingut uns resultats més rics ha estat canviar el conjunt d'imatges i entrenar la xarxa amb la llibreria *Flickr*, obtenint uns resultats que han sigut molt satisfactoris, ja que s'ha obtingut una bona resolució i nitidesa en les imatges.

S'ha de remarcar que, malgrat que els resultats han estat bons en la duplicació de les dimensions de les imatges, la implementació d'aquesta llibreria en una xarxa que quadrupliqués la resolució no ha obtingut tants bons resultats, ja que aquesta, en alguns casos, difumina subtilment la imatge final. És evident que es tracta d'un problema d'una dificultat més elevada, perquè, com que la resolució de les imatges d'entrada és de 64×64, en alguns casos és molt complicat distingir a simple vista els elements que les componen. Tot i això, seria interessant poder encadenar la sortida d'aquesta xarxa amb una altra que fes el procés d'enfocar la imatge. Encara que, segurament, si la xarxa que hem creat hagués estat entrenada amb un conjunt d'imatges encara major i amb l'ús de més epochs, cosa que implicaria un cost computacional immens, segurament hi hauria un increment en la definició de les imatges.

La super-resolució és una tècnica relativament recent, per tant, és molt probable que, entre el

desenvolupament tecnològic actual i els diferents descobriments que es van produint en aquest àmbit, s'arribi a perfeccionar aquesta tècnica i a donar uns resultats cada cop més captivadors. De fet, cal dir que en la contemporaneïtat d'aquest treball diversos investigadors de Google han dissenyat una xarxa neuronal convolucional que acompleix aquesta tasca i és capaç de donar uns resultats d'una qualitat gairebé al nivell de la imatge original (7).

En conclusió, és important remarcar que el *Deep Learning* és una eina increïblement capaç i versàtil que aconsegueix realitzar processos anteriorment impensables. La seva implementació en la super-resolució és, amb diferència, la millor solució al problema, si més no a dia d'avui.

Bibliografia

[1] CHAO DONG, CHEN CHANGE LOY, XIAOOU TANG *Accelerating the Super-Resolution Convolutional Neural Network*, 2016. Enllaç

[2] CHOLLET FRANÇOIS *Deep Learning with Python*, 2018.

[3] HAYKIN SIMON *Neural Networks: A Comprehensive Foundation*, 1994.

[4] P. KINGMA DIEDERIK, LEI BA JIMMY *Adam: A method for stochastic optimization*, 2014. Enllaç

[5] CHAO DONG, CHEN CHANGE LOY, KAIMING HE, XIAOOU TANG *Image Super-Resolution Using Deep Convolutional Networks*, 2015. Enllaç

[6] CHEN ZHEN, GUO XIAOQING,Y.M. WOO. PETER, YUAN YIXUAN *Super-Resolution Enhanced Medical Image Diagnosis with Sample Affinity Interaction* 2021

[7] SAHARIA CHITWAN, HO JONATHAN, CHAN WILLIAM, SALIMANS TIM, J. FLEET DAVID, NOROUZI MOHAMMAD *Image Super-Resolution via Iterative Refinement*, 2021.Enllaç

[8] GOODFELLOW IAN, BENGIO YOSHUA, COURVILLE AARON *Deep Learning*

[9] NIELSEN MICHAEL *Neural Networks and Deep Learning*, 2019.

[10] JASON BROWNLEE *Deep Learning Performance*, 2017.

[11] JASON BROWNLEE *Optimization*, 2021.