TREBALL DE FI DE GRAU

Grau en Estadística

Títol: Aprenentatge profund amb Keras

Autora: Júlia Tena Mensa

Director: Esteban Vegas i Ferran Reverter

Departament: Departament de Genètica, Microbiologia i Estadística de la Universitat de Barcelona

Convocatòria: Juny 2019

UNIVERSITAT DE BARCELONA

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat de Matemàtiques i Estadística

RESUM

Aquest treball parteix de la recerca bibliogràfica sobre la intel·ligència artificial, l'aprenentatge automàtic i l'aprenentatge profund. Per començar, en la introducció es troba síntesi de la informació consultada així com la definició i interpretació de la intel·ligència artificial, entre d'altres. En segon lloc, s'avalua la metodologia utilitzada, Seguidament, es treballen els conceptes teòrics que giren al voltant de l'aprenentatge automàtic i a continuació es desenvolupa la part pràctica del treball. En aquesta secció es troben diferents aplicacions d'aprenentatge automàtic sobre diferents camps, realitzant una breu descripció del tema, avaluant el codi aplicat i comentant els resultats. Per acabar, s'inclouen les conclusions del treball i la recerca bibliogràfica.

Paraules clau: aprenentatge automàtic, aprenentatge profund, Keras, capa neuronal artificial, backpropagation, optimitzadors, pesos, funció de pèrdua, entrenament.

Abstract

This work is based on bibliographic research on artificial intelligence, automatic learning and deep learning. To begin with, in the introduction there is a synthesis of the information consulted as well as the definition and interpretation of artificial intelligence, among others. Secondly, the methodology used is evaluated. Afterwards, the theoretical concepts that revolve around automatic learning are worked on and then the practical part of the work is developed. In this section you will find different automatic learning applications on different fields, making a brief description of the subject, evaluating the applied code and commenting on the results. Finally, the conclusions of the work and the bibliographic research are included.

Keywords: artificial intelligence, automatic learning, deep learning, Keras, artificial neural layer, backpropagation, optimizers, weights, loss function, training.

AMS: 97R40 - Artificial intelligence

ÍNDEX

ÍNDEX DE FIGURES

Gràfics

Taules

Il·lustracions

1. INTRODUCCIÓ

El gran tret que diferencia els humans de la resta d'essers vius és la seva capacitat de pensament i aquesta es converteix en una de les incògnites del raonament i la filosofia. Què és pensar? Què és aprendre? O què és ser intel·ligent? Al llarg de la història podem trobar diferents definicions que donen resposta a cada una d'aquestes preguntes, algunes extretes d'un procediment científic o un raonament lògic.

Així mateix, aquestes qüestions reben una importància rellevant a mitat del segle XX, quan es comença a parlar de la intel·ligència artificial. En aquest punt, no només ens podem qüestionar què es la intel·ligència o el raonament aplicat als éssers humans, sinó també a ordinadors. Alan Turing va donar les primeres respostes, establint un test per determinar si una màquina es pot considerar intel·ligent o no (Test de Turing[1]) i a partir d'aquest moment la intel·ligència artificial ha anat evolucionant, platejant noves preguntes i aportant respostes a cada nova invenció.

Dins d'aquesta evolució trobem que la intel·ligència artificial ha avançat en diferents direccions, adaptant-se a cada camp d'estudi: computació, finances, medicina, indústria... i a cada un d'ells, resolent els problemes amb innovadores aplicacions. Una d'aquestes és l'aprenentatge automàtic, i, en aquest cas, què és aprendre?

Aquesta és la pregunta que es vol respondre amb el present treball: què es considera aprendre en l'aprenentatge automàtic en el camp de l'aprenentatge profund. S'estudiaran els fonaments del *deep learning* amb les xarxes neuronals, definint els seus principis bàsics, estructures i funcionament. A partir d'aquí, es treballaran altres conceptes propis del *deep learning* per tal de poder obtenir un coneixement bàsic sobre el seu funcionament.

Així doncs, l'objectiu d'aquest treball, com en el *machine learning*, és aprendre: conèixer sobre que es basen aquestes noves tecnologies, com treballen i com aconsegueixen els resultats, molt d'ells, sorprenents.

El treball es dividirà en dues parts. La primera s'estudiaran els aspectes teòrics de les xarxes neuronals així com del *deep learning*, creant un coneixement pel treball pràctic del que s'ha estudiat anteriorment, per la segona part del treball. En aquesta part es treballaran diferents bases de dades sobre les qual s'hi aplicaran models d'aprenentatge profund, cada un d'ells amb un objectiu i temàtica diferent. En concret, es treballarà amb una base de dades genòmiques, imatges i text, amb els quals s'hauran d'aplicar funcions i models diferents. D'aquesta manera, es poden conèixer diferents aplicacions d'aquest camp del *machine learning*, com les funcions necessàries per desenvolupar-les.

[1] Turing, Alan. Computing Machinery and Intelligence. A: *Mind*. Regne Unit: 1950. vol. LIX, núm. 236. ISSN: 0026-4423

Es disposen de diferents eines per tractar problemes de xarxes neuronals, en aquest treball s'utilitzarà una biblioteca de Python anomenada Keras. Aquesta llibreria està especialitzada per treballar models de *deep learning,* oferint diferents funcionalitats per tractar dades variades i modelitzar xarxes neuronals concretes per cada tipus.

Per acabar, agrair als professors del treball, Esteban Vegas i Ferran Reverter per l'acompanyament i ajuda per realitzar aquest treball.

2. METODOLOGIA

El treball s'ha dividit en dues parts; la part teòrica, on s'aborden els diferents conceptes referents a la temàtica del treball i la part pràctica, on es treballen diferents aplicacions dels conceptes estudiats anteriorment.

En la part teòrica del treball s'han utilitzat recursos bibliogràfics com articles i llibres per estudiar els diferents apartats, partint d'una visió general i concretant en aquell àmbit d'interès.

Per la part pràctica, s'han realitzat tres aplicacions diferents a l'estudiat a l'apartat anterior. En les aplicacions realitzades es duen a terme passos semblants en cada una d'elles: importació de les dades com dels paquets necessaris, tractament d'aquestes, creació del model de *deep learning* i l'entrenament d'aquest i finalment avaluar la seva capacitat. Aquestes aplicacions s'han treballat amb el programari Keras escrit amb Python i executat sobre el *backend* TensorFlow. A més a més, durant el treball de les dades i dels models, també s'han utilitzat llibreries i funcions pròpies de Python. S'ha utilitzat el *software* Jupyter amb el qual s'ha pogut executar i veure els resultats del codi. Les dades que s'han utilitzat en les tres aplicacions s'han extret de Kaggle.

3. XARXES NEURONALS ARTIFICIALS

Les xarxes neuronals són un dels camps més estudiats i aplicats de la intel·ligència artificial, ja que aquestes permeten imitar l'aprenentatge humà en els ordinadors aconseguint, doncs, aproximar a l'objectiu de la intel·ligència artificial.

La idea base de les xarxes neuronal artificials és aconseguir emular el funcionament de les xarxes neuronals biològiques. Aquestes són un conjunt de cèl·lules, anomenades neurones, que es connecten entre si creant una xarxa amb la que, mitjançant impulsos nerviosos i treballant de forma paral·lela entre elles, poden dur a terme diverses activitats. D'entre aquestes activitats s'inclouen pensar, aprendre o memoritzar, tasques que des de fa temps diversos camps de la tecnologia i la informàtica volen incloure en el seus ordinadors.

La primera publicació de neurocomputació que es troben en la bibliografia es de 1943 en que McCulloch i Pitts van plantejar el primer model matemàtic inspirat en les neurones. A partir d'aquí, altres investigadors i científics de la època van seguir ampliant aquest camp d'estudi, per exemple, en el 1949 quan es va plantejar el primer mètode d'entrenament de xarxes neuronals artificials, anomenat regla de Hebb. Es van anar desenvolupant models matemàtics basats en la neurona biològica, obtenint així diferents estructures i algoritmes d'aprenentatge. Un fet destacable de la primera època de les xarxes neuronals artificials és que en el 1958 es va desenvolupar el primer neurocomputador on es desenvolupava el model de *Perceptron* i altres.

L'auge de la intel·ligència artificial es va veure interromput en l'any 1969 quan es va publicar *Perceptrons: An Introduction to Computation Geometry* de Minsky i Papert. En aquest estudi es parlava de les limitacions de les xarxes neuronals d'aquell moment, que només estaven compostes per una sola capa, i es demostrava com era impossible que una xarxa neuronal classifiqués patrons de classes separables no lineals.

No va ser fins després de 1980 que aquest camp va tornar a ser d'interès per científics, en el qual ja es disposaven d'ordinadors més potents, es coneixien algoritmes més robustos i eficients i la biologia havia avançat en el camp de la neurologia i el sistema nerviós. El llibre titulat *Parallel Distributed Processing* de Rumelhart, Hinton y Williams (1986) mostrava un algoritme (*backpropagation*) que permetia l'ajust de matrius de pesos de les xarxes amb més d'una capa. Amb aquesta publicació, les xarxes neuronals van recuperar el seu antic èxit, ampliant el coneixement i desenvolupant noves eines fins a dia d'avui.

3.1. Fonaments

Després d'aquesta breu introducció biològica i històrica, a continuació es treballaran els conceptes més tècnics de les xarxes neuronals artificials; per començar, què són i sobre què es basen.

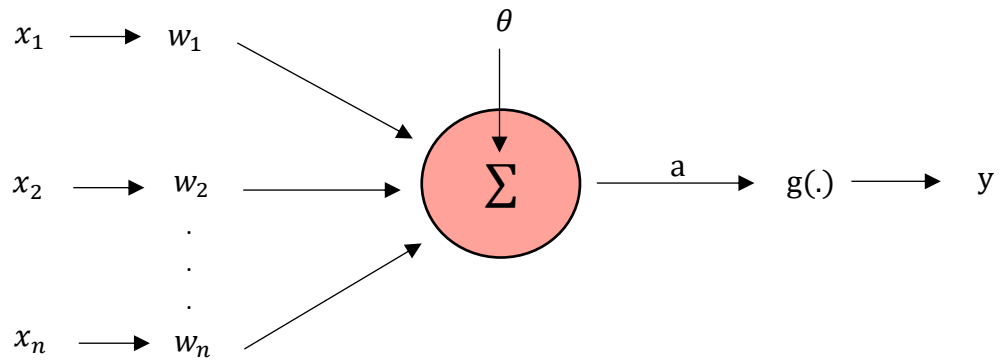
Les xarxes neuronals artificials són models computacionals inspirats en el sistema nerviós dels éssers vius, en el qual es consideren un conjunt d'unitats de processament, representades com a neurones, en que mitjançant connexions es poden enviar la informació entre elles.

Aquesta estructura serveix per la resolució de molts tipus de problemes, com per exemple, a partir d'una gran base de dades podem predir el valor d'una variable d'un individu en concret segons les altres variables d'aquell mateix individu o es pot arribar estimar el valor d'una casa segons tot el conjunt de variables que hi influeixen. No només problemes de predicció, sinó de classificació; per exemple, categoritzar els votants de diversos partits polítics.

Tot i que existeixen altres algoritmes i tècniques del *machine learning* que resolen aquestes mateixos problemes de predicció, les xarxes neuronals presenten unes característiques que permeten la resolució d'aquests amb molta més precisió que d'altres.

El model bàsic en el qual es van inspirar les xarxes neuronals artificials es representa en el següent gràfic:

Gràfic 1: Representació bàsica d'una xarxa neuronal



L'anterior gràfic representa la base dels model de les xarxes neuronals. A partir dels elements que s'hi poden observar en construeixen diferents tipus de xarxes neuronals amb les que es poden resoldre diferents problemes. Els elements bàsics que conformen tota xarxa neuronal artificial són (1):

- Valors d'entrada: Representants per la lletra *x* són els valors d'activació de la neurona. Aquestes valors es poden normalitzar per tal de millorar l'eficiència del algoritme.
- Pesos: Amb la lletra *w* es representen els pesos. Serveixen per quantificar la importància de cada variable.
- El valor de Σ representa l'operació lineal entre cada variable i el seu pes.
- La variable θ o *b* s'utilitza per representar el biaix de l'operació lineal.

- El valor *a* s'identifica com a *activation potential* i és la diferència entre l'operació lineal i el valor θ.
- La funció *g*, anomenada funció d'activació, és aquella que s'utilitza per enviar a través de les capes la informació.
- Les sortides *y* són els valors produïts per la xarxa neuronal a partir d'uns determinats valors d'entrada i el l'algoritme de desenvolupament de la xarxa neuronal.

En la majoria de xarxes neuronals, es pot veure com aquestes estan organitzades en capes, és per aquest motiu que en el seu tractament són necessaris els índex; s'indica amb un *k* el valor d'entrada a la neurona i *j* el valor de sortida i es pren *l* com l'índex de cada capa.

Amb el conjunt de paràmetres definits fins ara, es pot establir el següent càlcul:

$$a_j^l = g\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right)$$

on la suma és sobre totes les neurones de la capa anterior.

L'anterior expressió es pot reescriure per tal de fer els càlculs més manejables amb una matriu. D'aquesta manera, es defineix la matriu de pesos w^l, la matriu de biaixos b^l i el vector de valors d'activació a^l:

$$a^l = g(w^l a^{l-1} + b^l)$$

Dins aquests càlculs, també es computa la quantitat intermèdia de $z^l \equiv w^l a^{l-1} + b^l$, les entrades ponderades (*weighted input*). L'anterior equació també es pot expressar en termes de z^l:

$$z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l$$

$$a^l = g(z^l)$$

A part de ser un dels elements claus en la propagació de la informació al llarg de la xarxa, la importància de la funció d'activació també és deguda a dos motius. El primer d'aquest és que és un del components que intervé en el procés de *backpropagation*, que es comentarà en els propers apartats, i el segon d'ells és que introdueixen la no linealitat, característica que permet resoldre problemes que altres tècniques lineals no poden resoldre.

A partir de la funció *g(.)*, les xarxes neuronals artificials es poden classificaren dos grups a partir de la funció d'activació. Existeixen les funcions d'activació parcialment diferenciables (*partially differentiable activation functions*), que són aquelles en les quals hi ha punts on la derivada no existeix. Dins d'aquestes funcions es poden diferenciar la *step function*, *bipolar step function* i *symmetric ramp function*. En la *step function*, el valor de la resposta de *g(z)* seguirà el següent esquema:

$$g(z) = \begin{cases} 1 \; si \; z \geq 0 \\ 0 \; si \; z < 0 \end{cases}$$

En el cas de la *bipolar step function*, l'esquema és:

$$g(z) = \begin{cases} 1 \; si \; z > 0 \\ 0 \; si \; z = 0 \\ -1 \; si \; z < 0 \end{cases}$$

En alguns casos de xarxes, aquest esquema provoca errors en la classificació, és per això que es consideren les següents alternatives:

$$g(z) = \begin{cases} 1 \; si \; z \geq 0 \\ -1 \; si \; z < 0 \end{cases} \text{ o } g(z) = \begin{cases} 1 \; si \; z > 0 \\ z \; si \; z = 0 \\ -1 \; si \; z < 0 \end{cases}$$

Finalment, la *simmetryc ramp function* calcula el valor de g(z) a partir de un valor a que defineix el rang de la resposta.

$$g(z) = \begin{cases} p \; si \; z > p \\ z \; si \; -p \leq z \leq p \\ -p \; si \; z < p \end{cases}$$

On *z* es troba dins del rang *[-p,p]*.

Per altra banda, també existeixen les funcions d'activació totalment diferenciables (*fully differentiable activation functions*), que es caracteritzen per ser derivables en tots els punts del seu recorregut. Les quatre funcions més utilitzades en les xarxes neuronals són:

- Funció logística: $g(z) = \frac{1}{1+e^{-\beta * z}}$
- Funció de tangent hiperbòlica: $g(z) = \frac{1-e^{-\beta * z}}{1+e^{-\beta * z}}$
- Funció gaussiana: $g(z) = e^{-\frac{(z-c)^2}{2\sigma^2}}$
- Funció lineal: $g(z) = z$

Segons el tipus de resposta de la xanxa neuronal i com es plantegi el problema, es poden tenir en compte totes les funcions anteriors per *g(z)*.

Les funcions d'activació més importants són: (2)

- *Sigmoid* o la funció logística. Aquesta funció es caracteritza per que petits canvis en la variable *x* provoquen grans canvis a la *y* degut a la seva forma. S'utilitza principalment per problemes de classificació binària, on els resultats que s'obtenen es troben entre 0 i 1.

- *Tanh* o funció tangent hiperbòlica. Aquesta funció és semblant a l'anterior, però modificada de forma matemàtica. Els valors de sortida es troben entre -1 i 1 i s'utilitza en les capes ocultes de la xarxa neuronal. D'aquesta manera, com que la mitjana dels resultats són 0, la funció ajuda a centrar els valors i fa que l'aprenentatge a la següents capes sigui més fàcil (3).
- *ReLu*: Aquesta funció retorna el valor x si el valor és positiu i 0 si és negatiu. És la menys costosa de totes computacionalment perquè les seves operacions matemàtiques són més simples. A més a més, només algunes neurones s'activaran el que fa que la xarxa sigui més escassa i eficient.
- *Softmax function:* Aquesta funció és un tipus de funció logística i és molt útil en problemes de classificació de múltiples classes: retorna un valor entre 0 i 1 per cada una de la classes. S'acostuma a utilitzar en l'última capa, obtenint una probabilitat de classificació de cada classe.

És important fer una bona elecció per la funció d'activació que s'utilitzarà en cada capa ja que aquestes afecten a l'eficiència del model i als resultats.

3.2. Estructura

A partir d'aquest plantejament inicial de les xarxes neuronals i dels seus càlculs bàsics es poden establir diferents estructures pel seu funcionament. L'estructura de les xarxes neuronals defineixen les formes en que es connecten cada neurona i com es consideren els pesos en cada una d'elles. Les neurones es troben repartides en capes, un processador de dades que agafa un valor d'entrada i dona com a resultat un altre valor. Dins de les capes es troben els pesos de les xarxes, estructurats amb *tensors.*

En una qualsevol tipus d'estructura es mostren 3 elements (1):

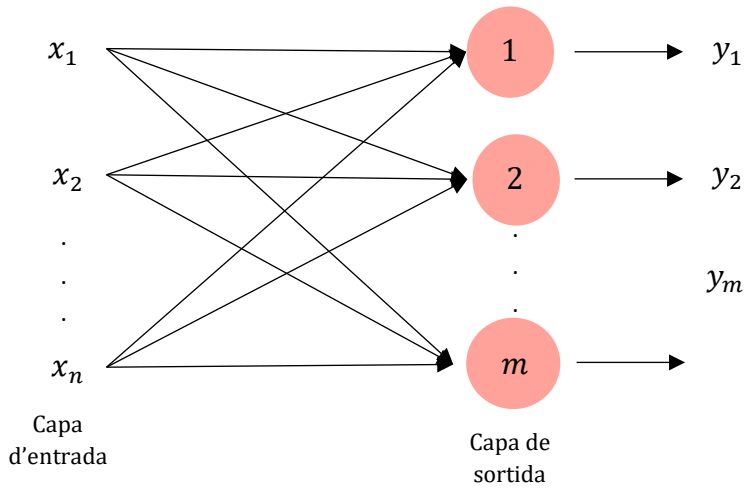
- Capa d'entrada: La capa d'entrada rep la informació de l'exterior. Molt sovint, aquesta informació es normalitza respecte la funció *g(z)* per tal de guanyar precisió matemàtica en les operacions que es duen a terme en l'algoritme.
- Capes intermèdies, ocultes o invisibles: Aquestes capes s'ocupen d'extreure informació que s'ha proporcionat de la capa d'entrada i trobar diferents característiques per la classificació o predicció resultant.
- Capa de sortida: És la capa on es mostra el resultat final de la xarxa neuronal. En xarxes neuronals connectades, aquesta capa pot significar la capa d'entrada a la següent xarxa neuronal.

Combinant aquestes tres capes principals amb diferents formes de connectar-les entre elles, s'obtenen 3 tipus d'estructures diferents.

3.2.1. *Single-layer feedforward architecture*

En aquesta estructura de xarxes neuronals només la composen la capa d'entrada i la capa de sortida. En aquest tipus, el nombre de neurones de sortida coincideix amb el nombre de paràmetres d'entrada on la informació es mou només cap a una direcció i s'utilitza principalment per problemes de filtratge lineal o de classificació (4).

Gràfic 2: Representació d'una arquitectura single-layer feedforward



Uns dels exemples d'algoritme on es segueix aquest exemple és el *Perceptron* i l'*ADALINE*, bastats en la regla de Hebb i la regla delta.

3.2.2. *Multiple-layer feedforward architectures*

A diferencia de l'anterior, aquesta estructura conté una o més d'una capa oculta, utilitzades en la resolució de diferents problemes, com per exemple, classificació, identificació, control, etc.

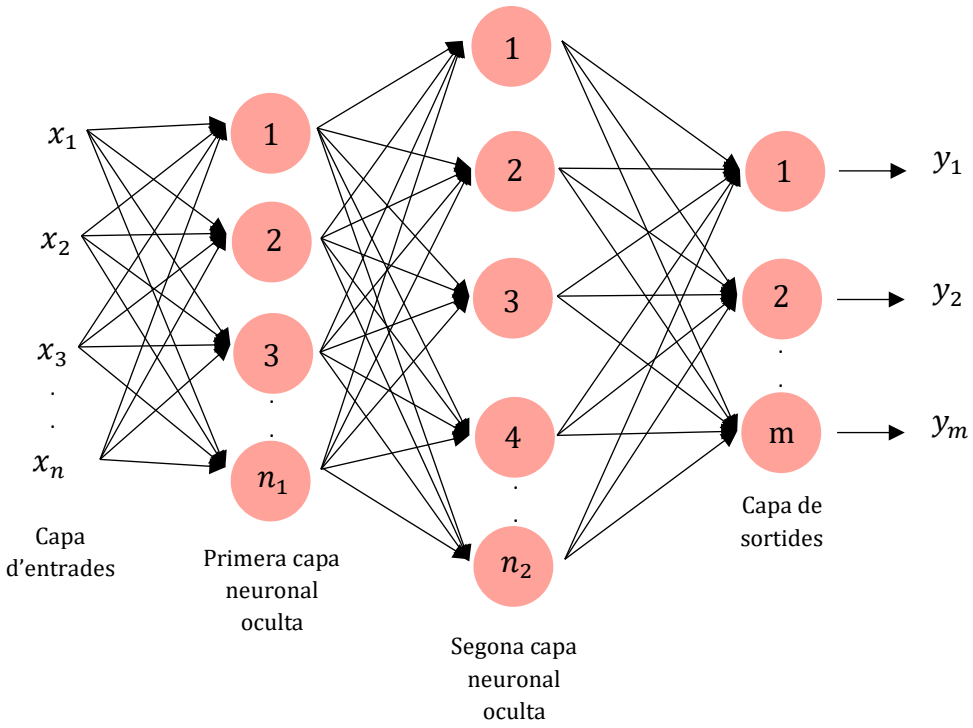
En aquest cas, la xarxa que es mostra està composta per la capa d'entrada i dues capes ocultes, amb n_1 i n_2 neurones cada una i finalment la capa de sortida, amb m neurones (les m possibles respostes de la xarxa).

La determinació del nombre de capes ocultes i del nombre de les neurones que la composen es molt variable; depèn de la qualitat i la quantitat de dades que es disposen i de la complexitat del problema, és per això, que en l'execució de de la xarxa neuronal, s'haurà de buscar la manera per tal d'aproximar-se de la forma més òptima a la resposta observada. No obstant, la capa de sortida sempre tindrà el mateix nombre de neurones que el possible nombre de respostes de la xarxa.

D'entre els models més populars que utilitzen aquest tipus d'estructura es troben el *radial basis function* (RBF) i la *multilayer perceptron* (MLP).

A continuació es mostra un esquema genèric d'aquesta estructura:

Gràfic 3: Representació d'una arquitectura múltiple-layer feedforward

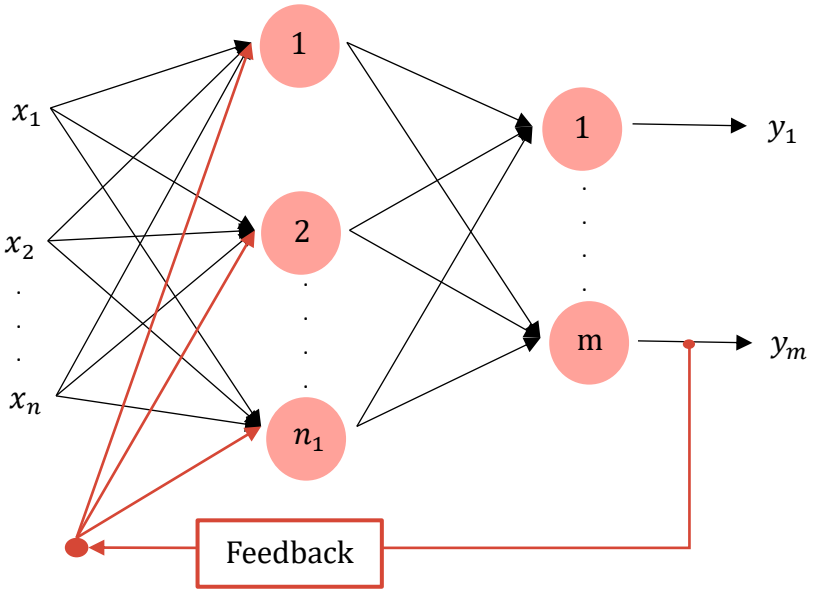


3.2.3. *Recurrent or feedback architecture*

La característica principal d'aquest tipus d'estructura és que, tal i com s'ha comentat anteriorment, el resultats de la capa de sortida serveixen per alimentar altres capes.

A continuació es mostra un exemple d'aquest tipus d'estructura on els resultats obtinguts serveixen per aportar nova informació a la cap oculta intermèdia. Amb aquest exemple de xarxa, la informació més antiga s'utilitzaria juntament amb la nova pel resultat definitiu de la xarxa.

Gràfic 4: Representació d'una arquitectura recurrent



També existeixen altres tipus d'estructures, per exemple, les *mesh architectures*, caracteritzada per l'ordre espacial de les neurones i de les capes. En aquest tipus d'arquitectures l'organització de la xarxa està relacionada amb el procés d'ajust dels pesos. S'utilitza per problemes de classificació, reconeixement de patrons, grafs i altres.

Tot i que hi ha tres estructures principals segons com estan connectades les capes, aquestes estructures poden combinar entre si per adaptar el model a les dades, per exemple, una xarxa neuronal aplicada a una base de dades formada per imatges, necessitarà una estructura *multiple-layer* i posteriorment se li podria afegir una capa recurrent. El més important és tenir en compte que segons les dades que es disposen s'hauran de fer servir un tipus de capes o altra, i cada una tindrà una sortida diferent; s'haurà de tenir en compte la compatibilitat.

3.3. Funcionament

En general, el procés de funcionament de les xarxes neuronals artificials es divideix en dues parts: el procés d'entrament o aprenentatge i el procés de test. En el procés d'entrenament es prepara la xarxa calculant els pesos de cada neurona i tots aquells valors que s'utilitzaran per, posteriorment, realitzar el procés del test. En l'entrenament de la xarxa, aquesta calcularà el pesos de cada neurona i determinarà tots els valors de la xarxa per tal d'aprendre aquells patrons i característiques necessaris per classificar o predir resultats amb certa precisió (5).

Per dur a terme aquests dos processos del funcionament de de la xarxa, es dividirà la base de dades en dues:

- Grup d'entrenament: Està formada entre un 60% i un 90% de les dades totals per utilitzar en el procés d'aprenentatge.
- Grup de test: Formades per un 10% i un 40%, la resta de les dades que no s'hagin tractat anteriorment, s'utilitzaran per provar la xarxa neuronal i veure el seu nivell de predicció, és a dir, dur a terme un procés de validació

Cada cop que s'entrena la xarxa neuronal amb el grup de dades d'entrenament es diu que s'ha fet una *epoch*.

Les xarxes neuronals artificials també es poden classificar segons el procés d'entrenament o d'aprenentatge que s'ha realitzat.

3.3.1. Procés supervisat

Consisteix en aprendre a representar les dades d'entrada a categories conegudes, donat un conjunt d'exemples. La majoria d'exemples d'aplicacions de l'aprenentatge profund es troben en aquesta categoria de processament. Aquest obté els resultats desitjats a partir de saber si ha predit bé un valor o no. Mitjançant una taula on es mostren les

entrades i les sortides amb els atributs i els valors, la xarxa neuronal, a cada resultat que obtingui, ho compararà amb el resultat real que li mostra la taula per un mateix cas. D'aquesta manera l'algoritme anirà ajustant els pesos en funció si ha predit bé o malament el valor

En aquest cas, una xarxa es considera entrenada quan la discrepància entre els valors predits i els reals es troba dins d'un rang de valors acceptable (6).

3.3.2. Procés no supervisat

A diferència del procés supervisat, el no supervisat no requereix cap taula on es mostrin els valors reals de la resposta, és a dir, es vol reconèixer què es mostra en les dades sense l'ajuda de cap categoria. L'algoritme va ajustant els pesos i els valors necessaris de la xarxa a partir del reconeixement d'aquells grups de dades que són més semblants entre ells (6).

En aquests tipus de procés es pot determinar a priori quin és el nombre màxim de grups de dades (*clusters*) que es vol que detecti l'algoritme.

A vegades, els processos no supervisats són necessaris com a passos inicials ens processos supervisats, per la millor comprensió de les dades.

3.3.3. Autoaprenentatge

Aquest tipus de processament forma part del procés supervisat, però té unes característiques concretes. Aquest procés d'aprenentatge no té una categoria resposta definida en la base de dades per la qual es pot identificar cada exemple de les dades, sinó que a partir de les dades genera les seves categories.

Els *autoencoders* són un exemple conegut d'autoaprenentatge, on les categories generades són els mateixos inputs. De la mateixa manera, tractar de predir el un vídeo, donat els anteriors o la següent paraula d'un text, donant paraules anteriors, són exemples d'aquest tipus de procés. L'autoaprenentatge es pot reinterpretar com a aprenentatge supervisat o no supervisat, depenent de si es té en compte al mecanisme d'aprenentatge o al context de la seva aplicació.

3.3.4. Aprenentatge reforçat

L'aprenentatge reforçat és una millora de l'aprenentatge supervisat. Seguint la mateixa idea d'aprendre a partir de l'experiència, l'entrenament reforçat no només té en compte si s'ha equivocat en el valor predit, sinó que també calcula si s'ha equivocat molt o poc. D'aquesta manera, el reajustament dels pesos de cada neurona canvia respecte el procés supervisat (7).

3.3.5. Altres

També existeixen altres tipus de procés d'entrenament, com els anomenats *online* o *offline.* El procés *offline* calcula els pesos després de processar tot el conjunt de dades d'entrenament, ja que a cada pas d'ajust en té en compte un nombre d'errors observats. En el *offline*, es necessiten en el moment d'aprenentatge totes les dades que formen part d'aquest grup. Davant de situacions en que no es disposen de totes les dades en el moment, s'estableix el procés online en que es té en compte que hi hagi canvis en el comportament de les dades .

3.4. Validació de la resposta

Amb l'entrenament de la xarxa neuronal no és suficient per dir que una xarxa és bona o està preparada per predir nous valors. Per poder acceptar una xarxa neuronal com entrenada ha de passar per un procés de validació.

En el treball de les xarxes neuronals, l'objectiu és trobar aquella que tingui el millor o el més alt possible nivell de predicció respecte totes les opcions possibles de xarxes. Una forma senzilla per trobar la xarxa més òptima és avaluar l'error en les seves prediccions utilitzant dades independents a les que s'han utilitzant en l'entrenament, però que formen part del mateix grup de dades.

Part de les dades que formen part del grup d'entrenament s'han de guardat en un grup apart anomenat grup de validació. Durant el procés d'aprenentatge, la xarxa neuronal utilitzarà les dades d'entrenament per calcular els pesos i els paràmetres. Un cop acabat aquest procés, amb les dades de validació, farà una primera aproximació de l'error de predicció del model, tot millorant aspectes del model que no acabin d'ajustar bé, anomenats els *hiperparàmetres* (aquelles variables que corresponen a la configuració externa del model, del qual del seu valor no pot ser estimat per les dades i són especificats pel programador per ajustar el algoritme), que són, per exemple, el nombre de capes o la mida d'aquestes. La idea és que en el procés de validació es busqui una millor configuració de la xarxa en l'espai de paràmetres trobat en el procés d'entrenament. Amb el model resultant i amb la predicció de l'error amb les dades de validació, es podrà escollir el millor model, però aquesta haurà de passar l'avaluació amb les dades de test (8) (9).

En el procés d'entrenament de les xarxes és té en compte una funció d'error definida sobre les dades que s'estan utilitzant. Un procés habitual s'anomena *hold out* que consisteix en avaluar el model amb les mateixes dades d'entrenament, però aquest pot provocar sobreajustament (*overfitting*).

Les dades de test, tal i com es va explicar en l'apartat anterior, són una part de la base de dades total que s'utilitza per la xarxa neuronal, que representa entre el 10% i el 40% de les dades, amb la que es busca fer una valuació final de la xarxa neuronal. És

important que la xarxa neuronal no conegui aquestes dades, ja que s'aconseguirà una predicció de l'error sense sentit.

Així doncs, les definicions dels tres conjunts de dades que es requereixen per la formació total d'una xarxa neuronal són:

- Dades d'entrenament: Un conjunt de dades que s'utilitzen per l'aprenentatge, és a dir, ajustar els paràmetres de la xarxa.
- Dades de validació: Un conjunt de dades que serveixen per millorar els paràmetres de classificació; avalua la capacitat de predicció del model mentre que el millora.
- Dades de test: Un conjunt de dades que s'utilitzen únicament per avaluar el rendiment final de la xarxa.

El mètode descrit, el *hold-out* no resulta el millor per la validació amb poques dades i en cas que les dades del test no siguin representatives de les dades d'entrenament. Per resoldre aquest últim problema existeixen altres mètodes com el de *k-fold cross.* Aquest mètode divideix les dades en *K* particions de la mateixa mida; per cada partició *i,* entrena el model en les altres i l'avalua sobre aquesta i el resultat final serà la mitjana dels resultats que s'obtinguin (*K* resultats). Quan es disposen de poques dades, es pot utilitzar el mètode de *iterated k-fold,* que consisteix en aplicar el mètode *k-fold* diferents cops, barrejant les dades cada cop abans de dividir-les en *K* grups. El resultat que s'obtindrà serà una mitjana entre els *K* resultats obtinguts en cada iteració.

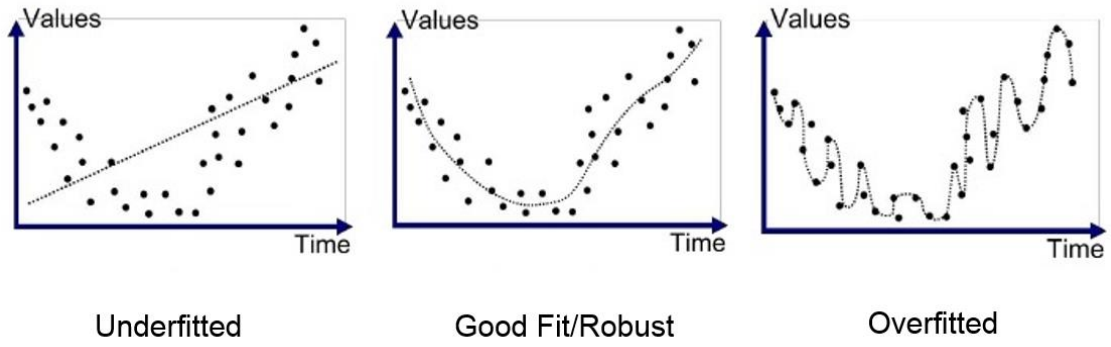
3.5. *Overfitting* i *underfitting*

Un cop realitzades un nombre d'iteracions en el procés de validació (*epochs*), l'ajustament aconseguit un cop arribat al punt òptim empitjora: el model comença ràpidament a sobreajustar les dades, és a dir, a predir massa bé cada un dels resultats. Aquest fet, anomenat *overfitting* és un dels problemes més comuns del de les xarxes neuronals i d'alguns algoritmes del *machine learning* (10).

L'objectiu és trobar aquell model capaç de fer un bon ajustament de les dades, però també que sigui capaç de predir dades que no ha treballat mai, és a dir, que sigui generalitzat per altres casos.

Al principi de l'entrenament, l'optimització i la generalització estan correlacionats: la pèrdua serà menor en les dades d'entrenament i en les de test. Mentre aquest fet es doni, es diu que el model està "sota ajustat" (*underfit*): el model encara no és capaç de predir totes les característiques de les dades. Al cap d'unes iteracions, la generalització deixa de millorar i les mesures de validació comencen a degradar-se, llavors el model passa a una situació de *overfit* on estima cada un dels punts que ha rebut a l'entrada. En aquests casos detecta patrons o característiques que no són significants.

Il·lustració 1: Representació de underfitting, bon ajust i overfitting



La millor solució per evitar que el *overfitting* és introduir noves dades d'entrenament, però si no és possible obtenir més dades, la següent solució és controlar la quantitat d'informació que s'emmagatzema o afegir restriccions sobre l'emmagatzematge de les dades, per exemple, afegir un paràmetre que sigui el nombre de característiques que ha d'aprendre la xarxa.

Aquest procés contra el sobreajustament s'anomena regularització (*regularitzation*) i a continuació es mostren algunes exemples d'aquest.

3.5.1. Reduir la mida de la xarxa

La forma més simple per prevenir el sobreajustament és reduir la mida del model, és a dir, el nombre de paràmetres que ha d'estimar.

El nombre de paràmetres que ha d'estimar un model pel seu aprenentatge es refereix com la seva capacitat, és a dir, un model que té més paràmetres tindrà més capacitat de memòria. Per una banda, es poden trobar alguns casos de models amb un nombre elevat de paràmetres per poder realitzar un ajustament de forma més bona (*a priori*) de les dades d'entrenament i per les dades del test. Aquest fet es dona sobretot en xarxes neuronals per problemes d'aprenentatge profund. Per altra banda, també existeixen xarxes que tenen un nombre limitat de paràmetres, per tant, serà més difícil trobar una bona estimació dels seus valors, ja que aquests hauran de concentrar més informació. Cal trobar un nombre de paràmetres que permeti tenir prou capacitat del model sense dificultar l'estimació dels valors.

3.5.2. Regularització dels pesos (weight regularization)

S'ha demostrat que els models de xarxes neuronals més simples tenen menys probabilitats de caure en el sobreajustament, per aquest motiu, una manera de suavitzar el *overfitting* és posar restriccions en la xarxa neuronal forçant els pesos a prendre valors petits, amb el que s'aconseguirà que la seva distribució sigui més regular. Aquest procés s'anomena regularització dels pesos, i es fa afegint a la funció de pèrdua de la xarxa el cost associat a tenir pesos amb grans valors.

3.5.3. Dropout

El *dropout* és una de les tècniques de regularització més comuna en les xarxes neuronals, desenvolupada per Geohh Hinton. Aquesta tècnica, aplicada a les capes, consisteix a abandonar de forma aleatòria (indicant amb un 0) diverses característiques de sortida de la capa durant l'entrenament, és a dir, convertir a 0 alguns dels paràmetres que formen la xarxa donada. Es calcula la taxa de *drop-out*, la fracció de paràmetres que s'han convertit a 0, normalment situada entre 0.2 i 0.5. En el moment del test, no hi ha unitats amb el valor 0 (introduïdes a través d'aquest mètode), sinó que es re-escalen els valors dels paràmetres de la xarxa a partir de la taxa de *dropout*.

Aquesta tècnica es basa en el fet de que introduint cert soroll en els resultats de la xarxa es podria evitar caure en característiques insignificants que es podrien tenir en compte.

4. APRENENTATGE PROFUND (*DEEP LEARNING*)

L'aprenentatge profund, en anglès conegut com *deep learning,* és una tècnica dins de les que composen el l'aprenentatge automàtic (*machine learning*), però es diferencia dels altres mètodes d'aquest camp de la intel·ligència artificial per basar-se en un funcionament més complex. Els algoritmes de *deep learning* es basen en un conjunt de capes on s'hi desenvolupen transformacions, és a dir, xarxes neuronals que en aquest camp poden estar formades per un gran nombre de capes (11).

A part de les diferents característiques del *deep learning* que s'estudiaran al llarg dels següents apartats, el punt més diferenciable d'aquest camp del *machine learning* amb els altres que el formen és que a mesura que es disposa de més dades l'ajust dels models milloren mentre que amb altres tècniques fora d'aquest camp, per molt que s'introdueixin més dades l'ajust no millora.

Tot i que des de 1989 que es disposen de les idees clau pel *deep learning* i l'algoritme fonamental pel seu desenvolupament des del 1997, el perquè ha sigut ara (des de 2012) que s'ha començat a desenvolupar aquest camp del *machine learning* és per motius tècnics; és ara quan s'han obtingut i es segueixen obtenint *hardwares* potents, avanços en els algoritmes i en els tractament de les dades. Des de llavors, l'aprenentatge profund no ha deixat de sorprendre aportant noves solucions a diferents problemes de diferents camps, introduint nous avanços i amb resultats cada cop millors.

Gràcies a tots els avanços tecnològics que l'acompanyen, fan que el *deep learning* sigui un dels mètodes més aplicats entre tots els àmbits del *machine learning*. A continuació es mostren alguns exemples d'aquestes aplicacions (12) (13):

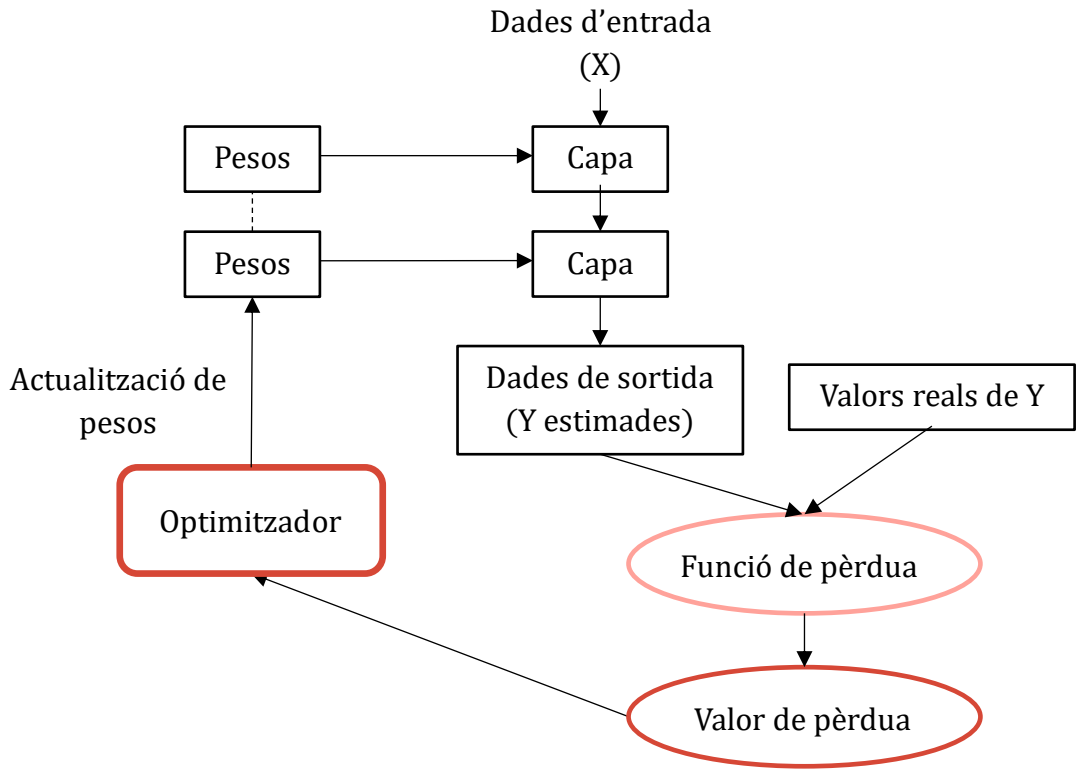
- En el camp de la bioinformàtica, mitjançant els *autoencoders*, estructures de xarxes neuronals, es pot avançar en el camp de la ontologia genètica i buscar relacions entre els gens i la seva funcionalitat.
- Empreses de màrqueting poden determinar el comportament dels clients a partir de detectar les característiques de cada un, predir el temps de vida de cada client i avançar-se a possibles accions.
- El reconeixement d'imatges és una de les aplicacions del *deep learning* més esteses. El seu funcionament permet detectar les característiques d'aquest tipus de dades i poder determinar que s'hi mostra. Aquest camp ha evolucionat fins al punt que també es poden determinar elements dins de la mateixa imatge, per exemple, molt utilitzat amb els cotxes de conducció autònoma, que poden identificar quins elements hi ha a la imatge com senyals de stop o persones.
- En el camp de la salut, s'utilitzen algoritmes de *machine learning* per detectar cèl·lules cancerígenes de forma automàtica.

No només s'apliquen les tècniques del *deep learning* per avançar en camps científics o tecnològics, sinó que també s'utilitzen per millorar el dia a dia dels éssers humans, per exemple, programes de música (*Spotify*) l'utilitzen per mostrar aquells grups que ens poden agradar o en els programes de traducció, com el *Google Translate*.

4.1. Funcionament general

Per entendre, de forma general, el funcionament dels models *de deep learning,* es pot observar el següent esquema:

Gràfic 5: Esquema de models deep learning



L'objectiu del *deep learning* i en molts casos del *machine learning* és que a partir d'unes entrades, com una imatge, poder representar (*mapping*) el que s'hi mostra, fent un procés d'aprenentatge i obtenir com a resultat una categoria, per exemple, la categoria "lleó". En aquest exemple, introduint a un algoritme d'aprenentatge profund una imatge d'un lleó, que ha aprés de moltes imatges d'animals que viuen en un parc natural, és capaç de detectar l'animal que hi ha a la fotografia i retornar un resultat amb la categoria "lleó". Amb aquest procés de relació data-categoria s'utilitzen les xarxes neuronals formades per diferents capes, on es realitzen les transformacions de les dades.

Les transformacions tenen l'objectiu de poder visualitzar la informació d'aquestes de forma més fàcil i poder detectar elements durant el procés del *deep learning*. En aquest procés, la qüestió és trobar representacions de les dades d'entrada que disminueixen la dificultat de la tasca que ha de realitzar el model. Per exemple, el cas d'un model de *machine learning* que hagi de detectar el color vermell en una imatge. Si aquesta imatge es transforma en format RGB seria més fàcil de detectar el color que en la mateixa imatge en blanc i negre. Així doncs, en aquest context de la intel·ligència artificial, aprendre és trobar el procés per tenir les dades millor representades.

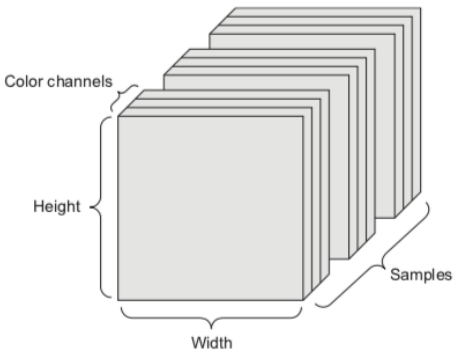
Dins dels models de *machine learning*, les transformacions que millor s'ajusten a les dades es busquen a través d'un espai d'operacions ja definit, anomenat espai d'hipòtesis, de forma que els algoritmes no són creatius en aquest aspecte. Aquí es troba una altra diferència entre el *machine learning* i el *deep learning*; el *deep learning* és capaç de trobar transformacions no definides a priori amb els pesos (*weights*). Els pesos són valors continguts dins de les capes que en funció dels valors que aquests prenguin es desenvolupa una transformació o altre. Tècnicament, la transformació implementada és parametritzada pels pesos. En *deep learning* "aprendre" és saber aquells pesos que permeten representar les dades i veure quina categoria estan associades.

Així doncs, l'objectiu del procés és trobar els valors dels pesos que mostrin les millors transformacions de les dades i aquest procediment es basa en la funció de pèrdua (*loss function)* o funció objectiu. La *loss function* avalua la diferència que hi ha entre els valors observats i els esperats de les sortides de la xarxa neuronal, de forma que s'obté una mesura sobre la predicció del model, el *loss score* o els valors de pèrdua. Amb aquest resultat final, es pot utilitzar com a coneixement per reajustar els pesos del model cap a la direcció amb la que s'obtenen resultats més bons, és a dir, minimitzar la funció de pèrdua.

4.2. Optimització del model

Per obtenir un resultat d'una model de *deep learning* s'haurà de fer tot un conjunt d'operacions algebraiques mitjançant els coneguts *tensors*, elements matemàtics que poden estar format per *n* dimensions, on s'emmagatzemen tots els paràmetres de la xarxa neuronal. Aquestes operacions es realitzaran entre els pesos de la xarxa i les dades disponibles.

Il·lustració 2: Representació de tensors d'una imatge



Tal i com s'ha comentat, quan s'aplica la xarxa neuronal es realitzen un conjunt d'operacions entre els pesos de de la xarxa, les dades i altres paràmetres que la conformen. També, el problema de la xarxa és determinar quins són aquells pesos que permeten aconseguir millors resultats. Arribats aquest punt es pot veure, doncs, que en el procés d'entrenament de les xarxes neuronals en els algoritmes de *deep learning* es calcularan els valors dels pesos. Inicialment, aquests valors són escollits aleatòriament de forma que el primer resultat que s'obtingui del *loss score* serà alt, però a mesura que la xarxa es vagi entrenant, els pesos s'aniran ajustant per obtenir cada cop millors resultats. Aquest entrenament segueix una sèrie de passos que es van aplicant de forma iterativa:

1. Agafa un conjunt de mostres de les dades d'entrenament amb els valors corresponents de la variable resposta.
2. Processa la xarxa amb les mostres que s'introdueixen i s'obté un valor de la variable resposta (l'observat).
3. Calcula el *loss score* del resultat en aquella mostra, és a dir, la diferència entre el valor de la resposta observada i esperada.
4. Actualitza els pesos de la xarxa en la mesura de reduir les diferències que s'han obtingut.

El procés anirà reproduint-se fins que la pèrdua es trobi dins d'un rang acceptable, en aquest moment, es podrà dir que la xarxa està entrenada. Els tres primers passos del algoritme es basen en operacions matricials mitjançant els *tensors*, però el quart pas, més complicat que els anteriors, es basa en la minimització de la funció de pèrdua per l'actualització dels pesos. D'aquesta manera s'obtindran uns valors que minimitzen la diferència entre els resultats obtinguts i esperats fins a aquell moment.

Per aquest procés són necessaris dos elements, la funció de pèrdua descrita anteriorment i els optimitzadors. Aquest últim elements és el que permet l'optimització de la funció de pèrdua i estableix un procés d'actualització dels paràmetres del model, que s'estudia a continuació.

4.2.1. Stochastic gradient descent

Donat una funció diferenciable, teòricament és possible trobar el mínim de la funció, és a dir, trobar aquells punts on la derivada és igual a 0. Així doncs, per obtenir el valor òptim cal trobar tots els punts on la derivada s'anul·la, avaluar-la amb la funció i escollir aquell punt que tingui un valor més petit. En les xarxes neuronals, caldria resoldre una equació formada per *n* variables (*n* paràmetres de la xarxa neuronal), habitualment, un valor no més petit que 1000. Com que el cost computacional de resoldre tal equació és molt alt, una forma eficient de trobar els paràmetres amb els que s'obtingui el mínim valor de la funció de pèrdua és modificar els paràmetres en funció de la direcció

obtinguda amb el gradient. D'aquesta manera, pas a pas, el model s'anirà acostant al punt òptim de la funció de pèrdua.

El procés que s'ha descrit s'anomena *mini-batch stochastic gradient descent* (mini-bach SGD) on en cada iteració del procés es pren un nombre aleatori de les dades d'entrenament, s'avaluen els valors dels paràmetres i a cada nova iteració, on s'introdueixen noves dades, s'actualitzen els paràmetres del model.

Aquest algoritme d'optimització, com en altres mètodes en xarxes neuronals, duu a terme la derivada de la funció de pèrdua amb una matriu denominada *W* i els valors d'entrada i de sortida. Combinant aquest resultats amb la derivada es podrà reduir el valor de la funció de pèrdua movent els valors de la matriu W en la direcció oposada als que marquen als gradient de la derivada de la funció; s'obtindran noves matrius W a través del càlcul $W_i -$ escalar $*$ gradient (f, W_i). El valor del escalar és important ja que el gradient només aproxima la curvatura quan s'està al voltant de W_i, per tant, aquest valor també haurà de ser petit (14).

A part del algoritme de SGD amb el *gradient descent* es poden trobar diferents variants en funció de com s'actualitzen els pesos de la xarxa a cada pas. Cal destacar, entre tots els opcions possibles, el concepte de *momentum* que tracta dos problemes: la convergència i els mínims locals. En el cas d'una funció de pèrdua que hi hagi un mínim local i un mínim global, si s'actualitza els pesos cap a la direcció del mínim local, observant els valors del seu voltant amb petits valor, es pot veure que la funció de pèrdua creix i que el mínim es trobarà en aquell punt. També existeixen altres optimitzadors més moderns, com l'*adam* o l'*adagrad*.

L'algoritme dona una metodologia i un procés per dur a terme l'optimització dels paràmetres a partir d'un gran nombre d'operacions matricials. Per resoldre-les cal implementar un mètode que permetés el càlcul de les operacions; aquest mètode s'anomena *backpropagation*.

4.2.2. Backpropagation

Per tal de calcular la derivada de la funció de pèrdua i modificar els valors de totes les matrius, com que estan connectades, es resol mitjançant la regla de la cadena, és a dir, a partir del valor de pèrdua trobat s'aniran modificant els pesos. Aquest procés, realitzat en un procediment d'actualització de paràmetres des de la darrera capa fins la primera, s'anomena *backpropagation*. El procés que es realitzen els càlculs des de l'entrada de dades fins l'obtenció del valor de pèrdua s'anomena *forward propagation* (15).

L'objectiu del procés de *backpropagation* és calcular les derivades parcial de $\frac{\partial C}{\partial w}$ i $\frac{\partial C}{\partial b}$ on C és la funció de pèrdua, *w* els pesos de cada capa i *b* el biaix. Els dos paràmetres *w* i *b* són els valors que es troben en l'expressió (aplicada a matrius) següent:

$$a^l = f(w^l \cdot a^{l-1} + b^l)$$

Aquesta expressió és equivalent a la que es va observar en l'anterior capítol: el valor de a^{l-1} és el que s'utilitza en càlcul de *z* ($z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l$)).

Per dur a terme el procés de *backpropagation* s'han de considerar dues hipòtesis sobre la forma de la funció de pèrdua:

1. La funció de pèrdua s'ha de poder escriure com una mitjana de totes les funcions de pèrdua fetes en l'entrenament. La raó de considerar aquesta hipòtesis és que inicialment es calcula per cada dades la funció de pèrdua i posteriorment s'utilitzaran els resultats per fer una mitjana.
2. La funció de pèrdua ha de poder ser escrita com a funció de sortides de la xarxa neuronal.

A partir d'aquestes dues hipòtesis, cal establir el valor δ_j^l: l'error que es dona a la neurona *j* de la capa *l,* que es calcula durant el procés de *backpropagation* i que a més a més està relacionat amb les derivades parcials de la funció de pèrdua.

En una xarxa neuronal, es poden observar els valors d'entrada a cada una de les capes. Si a aquesta valors se'ls introdueix una quantitat Δz_j^l, que s'anirà propagant al llarg de la xarxa i al final afectarà amb valor $\frac{\partial C}{\partial z_j^l}\Delta z_j^l$. Modificant aquesta quantitat es pot arribar a minimitzar la pèrdua; si $\frac{\partial C}{\partial z_j^l}$ és proper a 0 voldrà dir que la xarxa no millorarà gaire més modificant els valors d'entrada. Així doncs, es pot establir que:

$$\delta_j^l \equiv \frac{\partial C}{\partial z_j^l}$$

Es calcularan els valors de δ_j^l per finalment calcular els valors de les derivades parcials de la funció de pèrdua respecte els paràmetres del pesos i del biaix a partir de quatre equacions.

1. Equació per l'error de sortida δ^L:

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} g'(z_j^L)$$

   El primer element de la equació mesura com de ràpid canvia el valor de la funció de pèrdua en funció del valor d'activació per la neurona *j*. El segon terme mesura com de ràpid canvia la funció d'activació en z_j^L. En termes matricials, aquesta fórmula es pot reescriure com:

$$\delta^L = (a^L - y) \odot g'(z^l)$$

   On el producte dels dos elements es coneix com producte de Hadamard.

2. Equació per l'error de δ^l en termes de l'error de la següent capa:

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot g'(z^l)$$

El valor de $(w^{l+1})^T$ és el transposat de la matriu de pesos de la següent capa. Utilitzant els valors de la següent es vol "moure" l'error cap enrere, obtenint així una mesura de l'error. Es fa un producte de Hadamard amb els valors d'activació de la capa on es troba.

Combinant la primera equació i aquesta es pot calcular l'error de δ^l per qualsevol capa: primer s'utilitzarà la primera equació per calcular δ^L i a continuació es calcularà la segona equació per obtenir $\delta^{L-1}, \delta^{L-2}$...

3. Equació per la taxa de canvi del valor de pèrdua per qualsevol biaix de la xarxa:

$$\frac{\partial C}{\partial b_j^l} = \delta$$

4. Equació per la taxa de canvi del valor de pèrdua per qualsevol pes de la xarxa:

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^L$$

Aquesta equació mostra com calcular les derivades parcials de la funció de pèrdua respecte els pesos en termes de les quantitats de l'error de la capa *l* i els valors de *a.* Gràcies a aquesta equació, quan el valor de *a* és proper a 0 significarà que el valor de la derivada serà petit. En aquest cas es diu que la xarxa aprèn lentament.

Les equacions descrites es van modificant els pesos de la xarxa en funció dels valors de l'error que es va obtenint.

Actualment, i en xarxes futures, s'utilitzen processos més eficients i basats ens nous camps de treball capaços de fer diferenciació simbòlica, com el que s'aplicarà en la part pràctica d'aquest treball, anomenat *TensorFlow*. Aquest mètodes es basen en, donada una cadena d'operacions en que la seva derivada és coneguda, es pot calcular la funció gradient que assigna els valors dels paràmetres de la xarxa des d'aquesta. Així, quan s'haurien d'actualitzar els pesos, nomes caldria calcular sobre les noves dades la funció. Amb aquests mètodes moderns l'algoritme de *backpropagation* queda desplaçat.

4.3. *Convolutional Neural Networks* (CNN)

Després d'haver estudiat el comportament de les xarxes neuronals en el *deep learning*, hi ha un tipus de xarxa neuronal molt utilitzada en les aplicacions d'aquest, les *convolutional neural networks*, xarxes neuronals convolucionals. Aquest tipus de xarxa són utilitzades en tasques de visió per computadora i el reconeixent d'imatges, però també s'utilitza en altres camps, per exemple, a la biologia tractant amb dades genòmiques.

Les *convolutional neural networks* són similars a les xarxes vistes fins ara: estan formades per neurones que tenen diferents paràmetres i participen en el procés d'aprenentatge, però a diferència de les anteriors, aquest tipus de xarxes fan la suposició que les dades d'entrades tenen més d'una dimensió, d'aquesta manera, l'arquitectura de la xarxa estarà destinada a reconèixer elements concrets, per exemple, de les imatges (16).

El funcionament general de les xarxes neuronals convolucionals és que a cada capa s'identifica una característica de la imatge que s'ha introduït, de forma que a major nombre de capes major nivell d'abstracció d'informació. A diferència de les altres xarxes, aquest tipus conté dos capes que desenvolupen les dues operacions fonamentals: la convolució i el *pooling*.

Les capes convolucionals, que realitzen operacions de convolució, estan caracteritzades per aprendre patrons locals en petites zones de dues dimensions, és a dir, detectar característiques concretes de la imatge. La tasca a realitzar es basa en una sèrie de filtres (*kernels*) sobre un espai de dues dimensions o, en alguns casos, tres dimensions (considerant per exemple el color de la imatge). Aquesta propietat permet que una vegada apresa una característiques en un punt concret de la imatge, aquesta pot ser reconeguda en altres zones de la mateixa. Una segona característica d'aquest tipus de xarxa és que poden aprendre jerarquies espacials de patrons, per exemple, una primera capa convolucional pot aprendre elements bàsics i la segona pot aprendre patrons compostos formats pels elements bàsics anteriors.

S'acostumen a utilitzar capes de tipus *Dense* (totes les neurones de cada capa es connecten entre totes les de la següent capa) i que l'aprenentatge és de patrons globals en l'espai global d'entrada.

L'operació *pooling*, realitzada a les capes de *pooling*, és aquella que, acompanyant posteriorment a les capes convolucionals, realitzen una simplificació de la informació recollida per aquestes i creen una nova versió més compacta de la informació. Hi ha diferents formes de compactar la informació, per exemple, agafant el màxim valor (*max-pooling*) d'un grup de neurones de la capa convolucional i fent la mitjana d'aquells valors (*average-pooling*). L'objectiu d'aquesta tasca és reduir l'espai i el nombre de paràmetres, a part que ajuda a controlar el *overfitting* (17).

4.3.1. Funcionament de les capes convolucionals

En la seva definició, la convolució és una operació matemàtica en dos objectes per produir un resultat que expressa com la forma d'un és modificat per l'altre. Amb aquest càlcul, es pot detectar una característica particular dels valors d'entrada i obtenir el resultat amb informació d'aquesta característica. Els valors d'entrada s'anomenen *feature maps,* tensors de tres dimensions formats per l'altura, l'amplada i la profunditat. L'últim paràmetre correspon, per exemple, al color d'una imatge: en una imatge en RGB,

aquest valor prendrà 3 i una imatge en escala de grisos valdrà 1. Durant l'operació de convolució, s'extreuen aquestes capes i les aplica sobre les dades transformades, produint així el *output feature map*. Aquest serà un tensor de tres dimensions, però el valor de profunditat depèn del les capes. En aquest sentit, en el *output feature map* el valor de profunditat correspon als filtres.

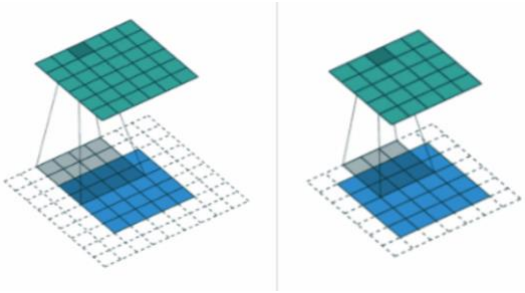
Així doncs, les capes de convolució estan definides per dos paràmetres principals:

- Mida de les capes de filtre, normalment 3x3 o 5x5.
- Profunditat de *feature map* de sortida: la quantitat de filtres calculats durant la operació.

El mapa de característiques que s'obté mitjançant les capes de convolució té dimensions de alçada i amplada diferents als valors d'entrada. Això és donat a les vores i els passos que realitza la capa de filtre al voltant de les entrades.

Si es vol conservar el mateix nombre de dimensions de les dades d'entrada per les de sortida, s'aplica el *padding*: afegir un nombre de columnes o files en cada entrada del mapa de característiques inicial. A més a més, d'aquesta manera no es perd informació que es troba en límits de la imatge.

Il·lustració 3: Representació del procés de padding

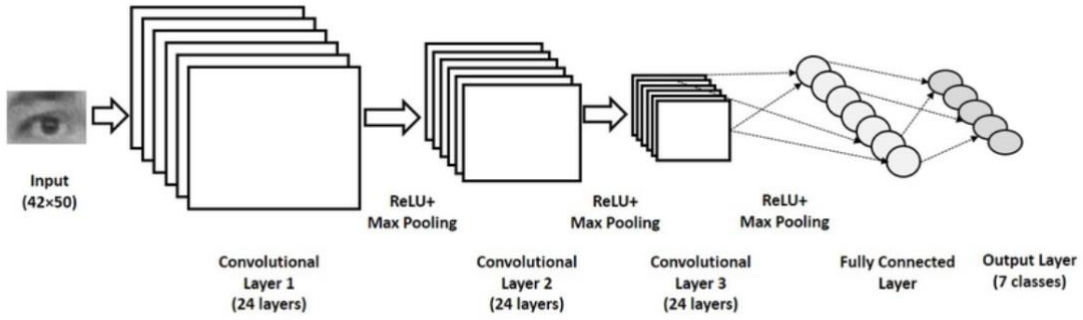


Un altre factor que por influir en les dimensions de sortida és el nombre de passos que realitza la capa de filtre per moure's a través de les dades d'entrada. Per defecte, aquest valor és 1, però a vegades es pot utilitzar valors més grans per reduir la mida de dimensions. Per aquest objectiu, s'acostuma a utilitza el *pooling*.

Una vegada feta l'operació de convolució, la informació passa a una següent capa on es realitza l'operació de *pooling*, aplicant el màxim o la mitjana. Aquesta operació acaba sent semblant a la que es produeix en les capes de convolució, però en comptes de realitzar una operació lineal es selecciona el màxim. Amb diverses capes de convolució i de *pooling* s'obtindran unes dades que hauran de redimensionar-se per ser acceptades en el model, de forma que en la part finar del procés, s'afegeix l'operació de *flattening* amb la que es pot convertit les dades en una matriu unidimensional per ser introduïdes a la següent capa, és a dir, "s'aplana" les sortides de les capes convolucionals convertint-les en vectors. Aquesta última tasca és necessària per l'estructura i el funcionament de la xarxa.

En resum, estaria representada com:

Il·lustració 4: Esquema d'una CNN



4.4. *Recurrent neural networks* (RNN)

Les estructures bàsiques de les xarxes neuronals segueixen un cicle en una sola direcció, cap endavant, on les dades inicial passen per totes les capes neuronals fins la sortida; aquest procés s'anomena *feedforward*. En el moment que s'afluixa aquesta condició, es poden permetre connexions cícliques dins de la xarxa amb el que s'obtenen xarxes neuronals recurrents (*recurrent neural networks*). D'aquesta manera, en les primeres xarxes neuronals, cada cop que processa un nou exemple, ha de començar de nou mentre les xarxes neuronals recurrents utilitzen la informació del passat pel seu processament.

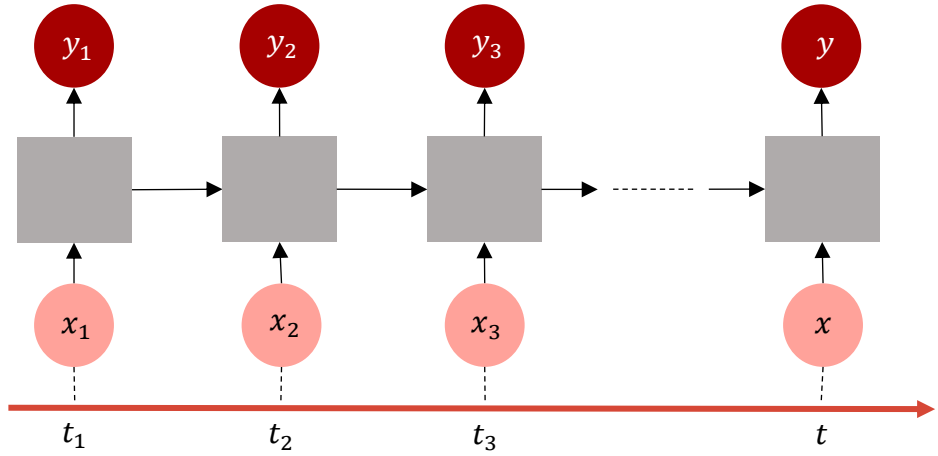
Aquesta estructura està adaptada a rebre dades seqüencials, és a dir, que totes les dades no s'ingressen al mateix temps, sinó que es van rebent durant el temps; hi ha un ordre en el temps. D'aquesta manera, l'aprenentatge de dades de nova entrada està complementat amb l'aprenentatge fet anteriorment gràcies a que es poden fer transferències d'informació cícliques.

Això permet que les RNN siguin un tipus de xarxa molt adequades per treballar problemes de reconeixement de veu, modelització del llenguatge o traducció, per exemple, en un text, quan aparegui la lletra *q* l'algoritme podrà inferir que la següent lletra sigui una *u,* o en sèrie temporals o seqüències de ADN. A més a més, aquesta característica no es pot trobar en altres tipus de capes, com en les CNN que la seva estructura està preparada per rebre altres tipus de dades.

Semblant al comportament humà, permetre que la informació del passat es mantingui en el procés (en les capes ocultes) fa que l'aprenentatge tingui memòria.

Aquesta xarxa està formada per la següent estructura:

Gràfic 6: Estructura d'una RNN



4.4.1. Funcionament capes neuronals recurrents

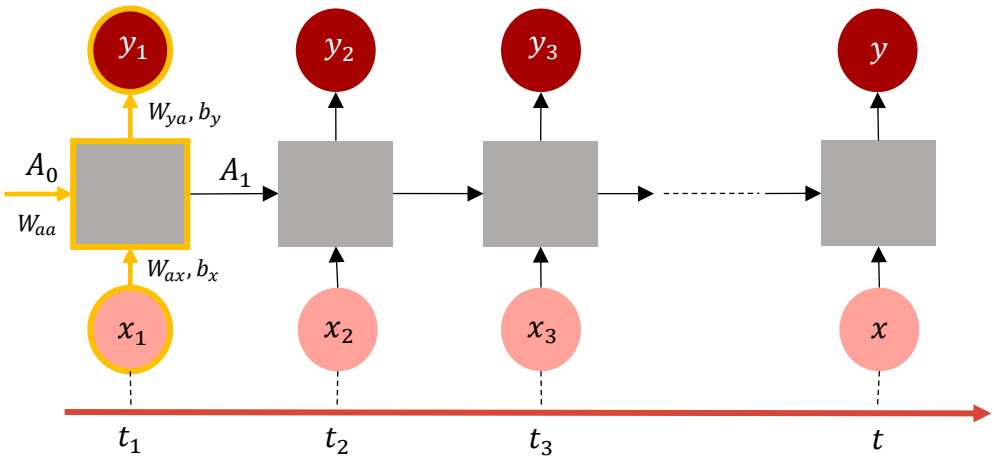
En la primera entrada, els càlculs que es realitzaran són:

$$Z_1 = W_{ax} * X_1 + W_{aa} * A_0 + b_a$$

$$A_1 = g(Z_1)$$

$$Y_1 = g(W_{ya} * A_1 + b_y)$$

Gràfic 7: Representació dels càlculs d'una RNN



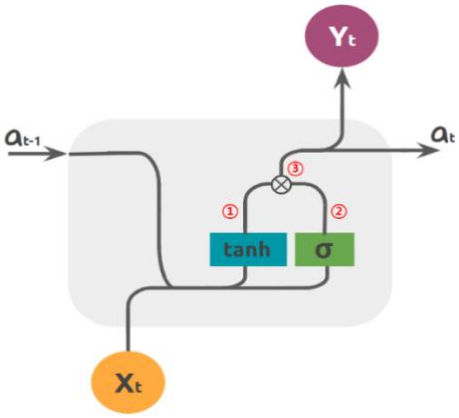
La informació que es manté oculta en el moment t es troba en Z_t. Aquest valor es calcula a través del valors d'entrada de x_t, modificats a través de la matriu W sumant l'estat ocult anterior i multiplicat per una matriu A, anomenada matriu de transició, formada per informació apresa anteriorment. Aquestes dues matrius contenen els pesos que

determinen la importància que se li dona a cada valor. Aquest resultat està aplicat a una funció que depèn de com es volen treballar els valors que s'obtinguin a part d'aconseguir que els gradients siguin factibles per dur a terme el procés de *backpropagation*. Amb aquesta estructura, no només s'utilitza informació del pas anterior, sinó que es van acumulant en els valors de la matriu A.

Quan es realitza el procés de *backpropagation* es van actualitzant el pesos de cada capa en funció dels gradients que s'han obtingut en la derivada de la funció de pèrdua. En el cas de les RNN, aquest procés també utilitza la variable resposta esperada, a part de l'observada. En el cas que els valors dels gradient siguin molt més gran que 1, es tindrien valors massa grans per utilitzar-los en la optimització del procés (*exploding gradients*), tot i que això no és un problema important ja que es podria modificar el rang de valors, però quan els valors dels gradient són més petits que 1, els resultats de l'actualització es van fent cada cops més petits i després d'algunes iteracions, no hi haurà diferències en els resultats (*vanishing gradients*). Per aquest motiu, en les xarxes neuronals recurrents, tenir un gran nombre de dades no és sinònim de que s'obtinguin resultats més bons.

Per solucionar aquest problema es disposa del procés de *Gated Recurrent Units* (GRU) on s'incorpora un nou element que guarda la informació més important per tenir en compte en l'optimització del model anomenat *gate controller (σ).* Aquest element dona un valor entre 0 i 1 (ja que es calcula a partir de la funció *sigmoid*) que participarà en el càlcul del valor de de la Y estimada, de forma que si el valor de la *gate controller* és proper a 1, aquesta informació resultarà important pel nou càlcul de y_t, i si és proper a 0 no ho serà (18).

Il·lustració 5: Representació de GRU



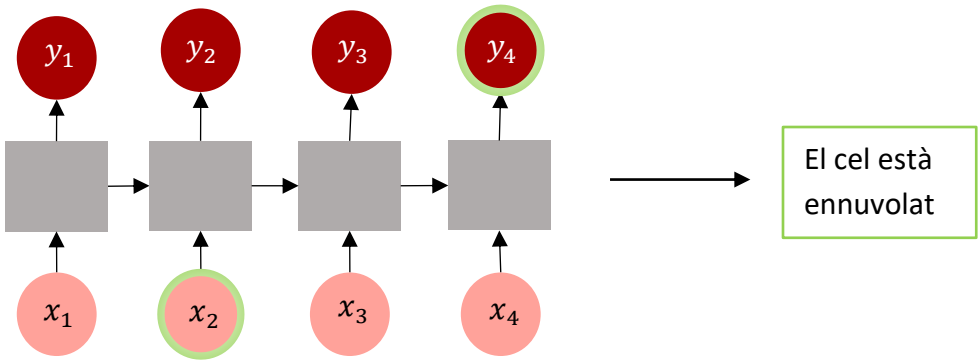
$$\hat{C}_t = \phi(W_a * [A_{t-1}, X_t] + b_a)$$
$$r_t = \sigma(W_r * [A_{t-1}, X_t] + b_{r)}$$
$$A_t = r_t * \hat{C}_t + (1 - r_t) * A_{t-1}$$
$$Y_t = g(W_y * A_t + b_a)$$

4.4.2. Long Short-Term Memory (LSTM)

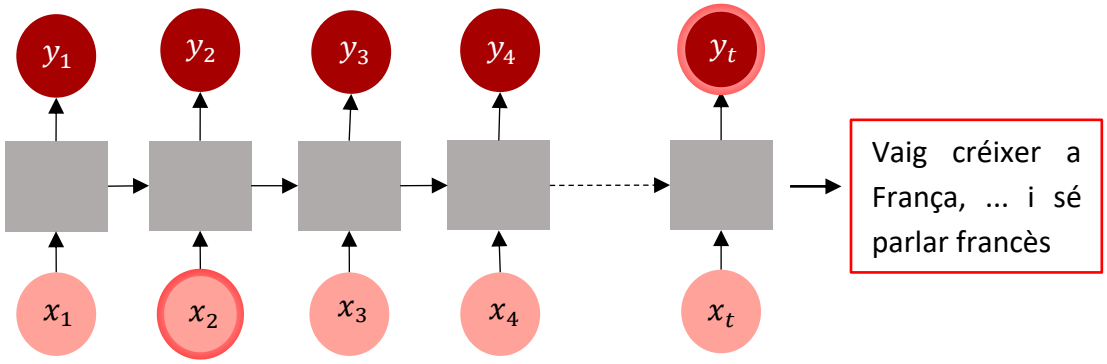
Les xarxes neuronals LSTM són un tipus de xarxes neuronals recurrents que segons la tasca que s'hagi de desenvolupar tenen resultats molt més bons i funcionen de forma molt eficient.

En un text format per "El cel està *ennuvolat*" és fàcil poder predir per una xarxa neuronal que *ennuvolat* és la següent paraula, en aquest casos la informació rellevant és pròxima a la paraula per predir, però quan es donen casos on la informació rellevant per predir una paraula es troba lluny de tal paraula, l'algoritme no acostuma a funcionar, per exemple, "Vaig créixer a França, tinc 18 anys, el meu pare és americà, la meva mare alemana i sé parlar *francès*".

Gràfic 8: Representació d'una bona predicció



Gràfic 9: Representació d'una predicció dolenta

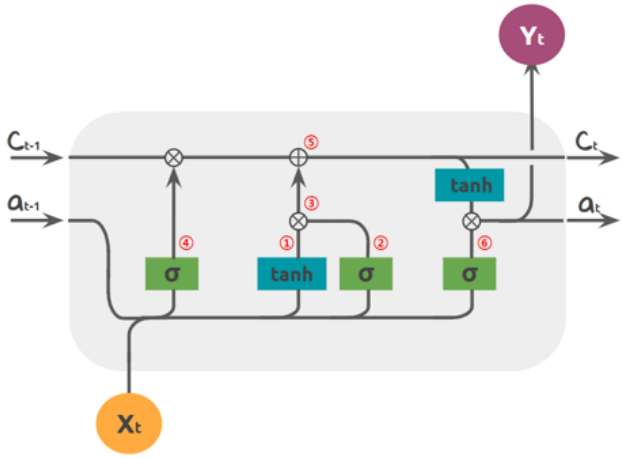


En el segon cas, s'utilitzen les xarxes neuronals *Long Short-Term Memory* (19). Estan dissenyades per recordar informació a llarg termini a partir de mantenir l'error al llarg del temps i de les capes. Al fer l'error més constant, permet que les xarxes neuronals continuïn aprenent al llarg el temps i de les iteracions, el que fa que es creï una manera de vincular causes i efectes.

El seu funcionament és semblant al procés de *Gated Recurrent Units*, però en aquest cas s'incorpora dues *gate controller* més: la *forget gate* i la *output gate*. A més a més, també incorpora un nou element anomenat *cell state* .

El següent gràfic mostra l'estructura que segueixen:

Il·lustració 6: Estructura d'una LSTM



5. KERAS

Keras és un entorn de treball de *deep learning* programat amb Python que proporciona una manera de definir i d'entrenar qualsevol tipus de model de *deep learning*. Aquest paquet va ser inicialment desenvolupat per investigadors amb l'objectiu de poder dur a terme un entrenament de xarxes més ràpid.

Keras es caracteritza per els següents punts:

- Permet utilitzar el mateix codi per ser executat sobre CPU o GPU.
- Compta amb una API fàcil que permet la creació ràpida de prototips de models.
- Incorpora un suport per xarxes convolucionals, xarxes recurrents i la combinació de les dues.
- Admet arquitectures de xarxes arbitràries: models d'entrada o sortida múltiple, ús compartit de capes, de models, etc. Això significa que Keras és molt útil per construir qualsevol model d'aprenentatge profund.

Keras desenvolupa models de *deep learning*, mentre que la part de la resolució d'operacions de tensors la realitza el seu *backend.* Actualment, Keras accepta tres tipus d'aplicacions de *backend*: TensorFlow, Theano i Microsoft Cognitive Toolkit. En les següents aplicacions de *deep learning* que es presenten en el treball, s'utilitzarà TensorFlow desenvolupat per Google (20).

Hi ha dues formes per definir el model: utilitzant la classe *sequential*, per conjunts de capes lineals o la funció API, per casos de de capes acícliques dirigides, que permeten construir arquitectures arbitràries. Una vegada definida l'arquitectura del model, els següents passos segueixen les mateixes funcions tant per si s'utilitza una funció *sequential* o API.

5.1. Pla de treball per problemes de *machine learning*

Per poder resoldre problemes de *machine learning* i, en aquest cas, de *deep learning*, es segueixen un conjunt de passos generals que, en la mesura del possible, s'intentaran aplicar en les posteriors aplicacions d'aquest treball (10).

Definició del problema

Per començar, cal plantejar quines dades es tenen i què es vol aconseguir amb elles. A més a més, també s'ha de tenir en compte quin tipus de problema es vol resoldre. Un cop contestades les preguntes que engloben el problema, al passar al següent pas, s'estaran acceptant dues hipòtesis:

- Els valors de sortida o les categories poden ser estimades a partir de les dades d'entrada.

- Les dades que es disposen són suficientment informatives per aprendre la relació que hi ha entre les dades d'entrada i els valors de sortida.

Fins que aquestes dues hipòtesis no es compleixin, hi haurà problemes que no es podran solucionar. Un d'aquests problemes és el *nonstationary problems*. Per exemple, es vol fer un manual de recomanació per la roba que es necessitarà durant el desembre i les dades que s'utilitzen són del mes d'agost. Per aquest tipus de problemes, amb dades cícliques, cal utilitzar dades anteriors, però pel mateix moment de temps. En general, cal tenir en compte que els models de *deep learning* només podrà predir aquells valors que ha entrenat.

Triar la mesura d'èxit

Abans d'entrenar l'algoritme, cal saber quina mesura s'utilitzarà per identificar si el model ajusta bé, a la vegada que aquesta s'utilitzarà per escollir la funció de pèrdua. Segons les dades que es tinguin, s'utilitzarà un tipus de mesura o altra.

Preparar les dades

El preprocessament de les dades o el *preprocessing* són aquell conjunt d'activitats destinades a convertir les dades a una forma més manejable per les xarxes neuronals. Aquestes activitats inclouen la vectorització, la normalització, el tractament dels valors perduts o *missings* i l'extracció de característiques:

- Vectorització: Tota estructura que contingui els valors de les categories en una xarxa neuronal ha d'estar en tensors de punt flotant o en casos concrets, en tensors d'enters. En tot cas, aquell procés de convertir les dades a tensors s'anomena vectorització de les dades.
- Normalització: Abans de començar a entrenar la xarxa neuronal amb el conjunt de dades, cal que les dades no tinguin valors "grans" en el sentit de nombre de dígits i que no hi hagi diferents rangs de valors. En definitiva, les dades han de prendre valors petits, normalment entre 0 i 1, i han de ser homogènies, i per aconseguir-ho la forma més efectiva és normalitzant la mitjana de cada variable per què sigui 0 i la desviació estàndard per què sigui 1, tot i que no sempre s'ha de fer (segons el tipus de problema).
- Tractament de *missings*: En el cas que 0 no sigui un valor de les variables de la base de dades les xarxes neuronals aprenen que aquest valor és un *missing* i l'ignoren en el seu estudi. Cal tenir en compte que si les dades d'entrenament no tenen cap missing i les dades de test si, la xarxa no reconeixerà què és un valor perdut i no ho sabrà detectar, és per això, que la presència d'un cert percentatge de *missings* és important pel bon funcionament de la xarxa. Tot i això, davant d'un problema elevat de dades perdudes, és recomanable tractar-les; el *deep learning* disposa de diferents algoritmes i mètodes que molts d'ells es sostenen sobre un fonament estadístic.

Si a més a més, existeixen problemes d'escassetat de dades, cal passar per un procés de *feature engineering.* El procés de *feature engineering* és aquell en que s'utilitza el propi coneixement de les dades i del algoritme de *deep* o *machine learning* que s'estigui treballant per tal d'obtenir un funcionament de l'algoritme més bo aplicant transformacions de les dades abans d'introduir-les al model. Per exemple, es vol crear un algoritme de xarxes neuronals que detecti l'hora que es representa en un dibuix d'un rellotge. Si s'introdueixen aquestes dades, l'algoritme tindrà un cost computacional molt alt per resoldre el problema, o inclús s'hauran d'utilitzar xarxes i models més complicats per un problema que si se li dur a terme un procés de *feature engineering* és senzill de resoldre. En aquest cas, es podria introduir una variable que expliqués les coordenades de les agulles del rellotge o l'angle que dibuixen. Introduint aquesta informació que per l'humà és molt fàcil de descriure, simplifiquem el problema de la xarxa i contribueix en el seus bons resultats.

Tot i que actualment hi ha algoritmes de *deep learning* que són capaços a de detectar característiques de les dades sense l'ajuda de l'humà, aquesta segueix sent una activitat important per dos motius: és una forma més elegant de resoldre els problemes i es necessita un nombre menor de dades.

Partint d'una línia de base

També cal tenir en compte sobre quina línia de base es parteix, és a dir, quin és el valor mínim de la mesura d'èxit que s'aconseguirà en el model, conegut com a *statistical power.* Aquest valor s'aconsegueix a través d'un petit model capaç de predir els resultats. Permet veure si es pot resoldre el problema que s'està plantejant, és a dir, si després de vàries arquitectures i models no s'ha aconseguit una predicció raonable serà perquè una de les dues hipòtesis que es plantegen no es compleix.

En el cas que es complissin, per fer el primer model cal tenir en compte tres elements:

- Activació de l'última capa: Aquest punt estableix unes restriccions útils pels resultats de la xarxa.
- Funció de pèrdua.
- Configuració de l'optimitzador.

Respecte la selecció de la funció de pèrdua, cal tenir en compte que no sempre és possible optimitzar directament la mesura d'èxit, ja que a vegades no és una tasca fàcil convertir-la en una funció de pèrdua. Per una banda, aquesta ha de donar resultat a un sol punt de les dades i ha de ser derivable. Per exemple, la mesura d'èxit de l'*accuracy* no es pot optimitzar directament

A la següent taula, es mostren les principals funcions de pèrdua i tipus d'activació per la última capa segons alguns dels problemes més comuns:

Taula 1: Principals funcions de pèrdua i tipus d'activació segons el problema

Problema	Funció d'activació de la última capa	Funció de pèrdua
Classificació binària	sigmoid	binary_crossentropy
Multiclass, classificació d'un nivel	Softmax	categorical_crossentropy
Multiclass, classificació amb nivells múltiples	Sigmoid	binary_crossentropy
Regressió (valors arbitràris)	NA	mse
Regressió (valors entre 0 i 1)	sigmoid	Mse/binary_crossentropy

Scaling up

Un cop s'ha obtingut un model amb prou *statistical power*, no se sap si aquest disposa de les característiques necessàries per resoldre tot tipus de problemes. Com s'ha comentat anteriorment, la qüestió principal en la formació de models de *deep learning* cau en trobar aquell punt que es trobi entre l'optimització i la generalització; entre la sub-capacitat i l'excés d'aquesta. Així doncs, per augmentar la generalització problema simplement es poden incloure capes al model, ampliar el nombre de neurones que la formen o entrenar el model amb més iteracions. Quan amb aquestes modificacions s'observi que la mesura de validació empitjora, s'haurà arribat al punt en que l'ajust sigui bo.

Regularitzant el model i ajustant els hiperparàmetres

En aquest pas del procés s'intenta arribar a aquell model més bo a partir de la modificació del paràmetres, del l'estructura del model, etc. Tasques que són importants per arribar a aquest objectiu són:

- El *dropout.*
- Provar diferents arquitectura: afegir o treure capes.
- Regularitzar els pesos de les capes.
- Provar diferents hiperparàmetres.

Cal tenir en compte que cada cop que s'avaluï el model sobre les dades de validació, aquest va extraient informació de les dades i del funcionament de l'avaluació, de forma que si es repeteix aquest procés moltes vegades els resultats que s'obtinguin poden no ser confiables.

Una vegada es té el model que, sobre les dades de validació i d'entrenament per separat, s'obtenen valors satisfactoris, es pot provar sobre el conjunt total de dades d'aquest dos grups, sense les dades de test. Si el valor de pèrdua que s'obtingui és pitjor que els que s'obtenia en l'entrenament o en la validació, voldrà dir que l'algoritme també ha après de les dades de validació. En aquest cas, s'hauria de modificar alguns paràmetres i canviar el procés de validació.

5.2. Funcions principals

En aquest apartat es descriuran algunes de les funcions principals que s'utilitzen per la resolució de problemes mitjançant l'entorn de Keras.

Paquets principals i funcions bàsiques

- *Numpy*: És una biblioteca que recull eines pel tractament d'objectes de matrius multidimensionals.
- *Panda*: És una llibreria que permet el tractament de bases de dades i el seu anàlisis.
- *Matplotlib*: És aquella biblioteca que permet dibuixar alguns gràfics per la representació del model o dels seus resultats.
- *Sklearn.preprocessing:* Tractament de variables.
- *Tensorflow.keras*: A partir d'aquest paquet es van carregant els diferents elements del model, com ara capes.
- *flow_from_directory():* Extracció d'informació de les carpetes

Tractament de les dades

- *ImageDataGenerator():* Utilitzada pel tractament d'imatges.
- *Tokenizer()*: Conjunt de funcions que permeten el tractament de dades de tipus caràcter, com paraules o textos.

Definició model

- *sequential()*: Funció inicial per construir una xarxa neuronal.
- *model.add():* una vegada definida la instrucció anterior, aquesta permet anar afegint capes a la xarxa i així anar construint el model.
- *model.compile()*: On s'especifiquen les mètriques per l'optimització del model.
- *model.summary()*: Resum del model creat.
- *model.fit():* Ajust del model.
- *fit_generator():* Ajust del model a partir d'un generador.
- *model.evaluate()*: Funció que permet avaluar el model entrenat.
- *evaluate_generator()*: Similar a *model.evaluate()* tractant dades a partir de generadors.
- *model.predict()*: Funció que permet calcular prediccions a partir del model entrenat i dades test.
- *predict_generator()*: Similar a *model.predict()* tractant dades a partir de generadors.
- *callback():* Paràmetres que permeten el millor ajust del model.

6. IMPLEMENTACIÓ DE MODELS AMB KERAS

Un cop treballades la part teòrica dels problemes de *deep learning* juntament amb les principals funcions del paquet Keras, en el següent apartat es treballaran diferents aplicacions en el qual es realitzaran models de *deep learning* amb les funcions estudiades.

Amb els problemes que es treballen a continuació tenen diferents tipologies de bases de dades i diferents plantejaments, de forma que dona una àmplia visió de les possibilitats que ofereix el paquet per treballar qualsevol tipus de dades.

6.1. Aplicació 1: "*A primer on deep learning in genomics*" (21)

6.1.1. Descripció de l'article

El primer problema que es resoldrà serà a partir de l'article "*A primer on deep learning in genòmics*".

Durant els primers punts de l'article, aquest fa una breu introducció al *deep learning*: repassa els conceptes de l'aprenentatge supervisat i no supervisat i els tipus de problemes que resolen, l'objectiu i el funcionament del models de *deep learning*, explicant els tipus de grups de dades que s'utilitzen per l'optimització. Al final d'aquesta introducció, els autors expliquen la diferència fonamental que hi ha entre les tècniques de l'aprenentatge automàtic i l'aprenentatge profund: l'abast de tractament de dades del segon és molt major i els seus sistemes d'optimització i d'estimació dels paràmetres permeten obtenir un nivell de predicció alt per diferents problemes.

Amb aquesta breu introducció al *deep learning*, detalla a continuació alguns aspectes de les xarxes neuronals, que serveixen de pont per comentar algunes de les arquitectures de xarxes neuronals per problemes d'aprenentatge supervisat que existeixen: la *feed-forward*, la *convolutional* i la *recurrent neural network*. Tal i com s'ha estudiat durant els capítols del treball, la primera de les arquitectures és aquella que permet la resolució de problemes de predicció normals, sense hipòtesis sobre les dades d'entrada i on les neurones de la capa i estan connectades entre totes les neurones de la capa i+1. En el segon tipus d'arquitectures, les CNN, les seves neurones estan destinades a analitzar certs patrons de les dades d'entrades i a partir d'aquí saber-los reconèixer en altres zones de les dades o en noves dades. Finalment, les xarxes neuronals recurrents són ideals per tractar seqüències de dades o dades temporals a partir d'elements que són capaços de memoritzar informació.

Seguidament es treballa l'optimització dels models a partir de la funció de pèrdua, calculada a través de l'error que resulta dels valors reals i els valors predits pels models. Com millor sigui el model, més petit serà l'error, per tant, per la bona predicció de valors de la xarxa cal fer que la funció de pèrdua obtingui valors mínims. A part d'aquesta

mesura de validació, també comenta els errors al quadrat i la *cross-entropy*, la diferència entre dos distribucions de probabilitat sobre el mateix grup de dades, habitualment utilitzada en dades categòriques.

Amb un petit comentari pel que fa el procés de *backpropagation* i com es combina la regla de la cadena amb la funció de pèrdua, es passa analitzar aquelles tècniques que fan més efectius els models: un bon *preprocessing* i una mètrica d'avaluació adequada. A part d'aquestes dues, també afavoreix els millor resultats del model si es té coneixement de les dades que s'estan tractant; segons el tema de les dades i el coneixement del seu comportament es podria dissenyar una arquitectura que s'adaptés millor a aquestes. Per exemple, les dades genòmiques acostumen a estar molt poc equilibrades, de forma que és millor avaluar el model a partir del *accuracy* o el *recall*. Tot i això, el que assegura un bon resultat en qualsevol tipus de model és disposar una gran base de dades per poder estimar tots els paràmetres, és per això que quan es té una base petita poden funcionar millor tècniques que treballin amb menys paràmetres.

Obtenir un bon resultat de predicció és important, però molts cops és més important pel fet de poder entendre la interpretació del model, sobretot en dades genòmiques. Per exemple, en el camp de la cromatografia, obtenir models amb molt bona predicció serveixen per aprendre una nova gramàtica de la regulació genètica.

Els autors de l'article presenten que les tècniques de *deep learning* cada cop són més utilitzades per estudiar el camp de la genètica, destacant el *functional genomic.* Per exemple, la predicció de seqüències de DNA o RNA, expressions de gens o controls d'unió. En aquests caps cal destacar els bons resultats que s'han obtingut en el camp de *regulatory genòmics,* a partir de l'ús d'arquitectures directament adaptades amb CNN o RNN.

Existeixen diferents eines adequades per extreure patrons en les seqüències de transcriptoma amb l'objectiu de trobar quant d'ARN es produeix a partir d'un tros d'ADN o una condició en particular; el *deep learning* pot construir models predictors d'expressió genètica a partir d'aquestes dades i poden ser utilitzats per l'estudi dels models de *splicing-code* com la identificació de RNA sense codificar.

A part dels camps comentats, la predicció de fenotips és una altra àrea d'interès pel *deep learning*. Un primer pas per realitzar aquest tipus de prediccions és especificar quin tipus de variants genètiques estan present en un sol genoma; aquest problema ha estat resolt a partir d'una xarxa CNN amb una imatge d'ADN alineat.

Tal i com s'ha pogut observar, el *deep learning* té molt de potencial en el camp de les dades genètiques, però encara fa falta molt més coneixement científic i tecnològic per abordar altres problemes com el disseny de de sistemes de *deep learning* que complementin la presa de decisions mèdiques o com evitar biaixos en el procés d'entrenament.

6.1.2. Resolució

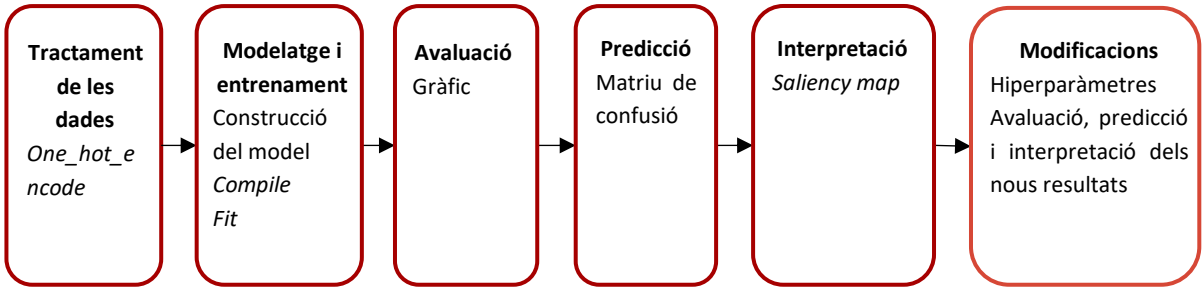
Aquest article inclou un exercici on es mostra com utilitzar un model de *deep learning* per aproximar un dels problemes en el camp de la genòmica funcional: descobrir els llocs d'unió del factor de transcripció de l'ADN. La genòmica funcional es el camp de la biologia molecular que a partir d'un gran nombre de dades que mostren gens i transcripcions per descobrir funcions i interaccions entre gens i proteïnes.

Així doncs, es dissenyarà una xarxa neuronal que pot predir les unions en l'ADN basant-se en els resultats d'uns assaig; en aquest cas les seqüències d'ADN més llargues seran les variables predictores mentre que la resposta positiva o negativa de l'assaig serà la variable de resposta. S'utilitzaran dades simulades que consisteixen en seqüències d'ADN de 50 bases de longitud i s'etiquetaran amb un 0 o un 1 segons el resultat. L'objectiu final és obtenir un classificador que pot predir si una seqüència particular unirà la proteïna i descobrir un *motifs* curt, un lloc d'unió de seqüències que estan unides a la proteïna.

Tot el codi del problema amb els respectius resultats es troben enumerats al Annex 1.

L'esquema que es seguirà durant la resolució del problema serà el següent:

Gràfic 10: Esquema de la resolució



Tractament de les dades

En l'objectiu d'entrenar la xarxa neuronal, s'ha de carregar la base de dades i dur a terme una sèrie de tècniques pel tractament de la base. Aquestes tècniques són necessàries perquè el model pugui aprendre de forma més eficient.

Per començar, cal carregar les llibreries que s'utilitzaran i la base de dades. En aquest cas, es carregen els paquets següents *Numpy, Panda Matplotlib* i *Requests*, que permet obtenir el contingut d'una URL.

El següent pas és organitzar la base de dades per tal de ser introduïda en el model a partir de tensors. En aquest cas, s'haurà de transformar la base de dades de forma que estigui organitzada en una matriu numèrica aplicant la transformació *one_hot_encoding*, una forma de transformar els diferents valors d'una variable a conjunt de valors que només utilitzen 0 i 1.

La base de dades la formen 2000 seqüències de 50 bases cada una. Primer de tot, s'indicarà cada base en un vector de 4 dimensions, aplicant la funció *OneHotEncoder* on hi haurà un 1 on es correspongui la base: quan hi hagi una A el vector serà [1,0,0,0], una C [0,1,0,0], la T amb [0,0,1,0] i finalment la G formada per [0,0,0,1]. Tot seguit es concatenen tots aquests vectors de 4 dimensions al llarg de la seqüència per formar una matriu.

En el codi següent es carrega un paquet anomenat *sklearn.preprocessing* que proporciona algunes funcions per tal de transformar classes i nivells de variables. A partir d'aquest paquet, s'importen les funcions de *LabelEncoder* i *OneHotEncoder*. El primer d'ells s'utilitza per convertir dades categòriques en numèriques a partir d'etiquetar cada valor categòric a un numèric i el següent, *OneHotEncoder*, passa aquesta valors numèrics en un vector de 4 dimensions, on cada fila correspondrà a una de les 4 posicions, tal i com es pot observar en el resultat.

Codi 1: Tractament de la base de dades

```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder

# Codifica una seqüència de bases com una seqüència de enters.
integer_encoder = LabelEncoder()
# converteix una matriu d'enters en una matriu escassa on
# cada fila correspon a un possible valor de cada funció.
one_hot_encoder = OneHotEncoder(categories='auto')
input_features = []

# Amb un for es va tranformant la base de dades
for sequence in sequences:
  integer_encoded = integer_encoder.fit_transform(list(sequence)
)
# Fit_tranform: Ajusta el codificador d'etiquetes i torna les
# etiquetes codificades
integer_encoded = np.array(integer_encoded).reshape(-1, 1)
one_hot_encoded = one_hot_encoder.fit_transform(integer_encoded)
input_features.append(one_hot_encoded.toarray())

np.set_printoptions(threshold=40)
input_features = np.stack(input_features)
print("Example sequence\n-----------------------")
print('DNA Sequence #1:\n',sequences[0][:10],'...',sequences[0][
-10:])
print('One hot encoding of Sequence #1:\n',input_features[0].T)
```

El resultat que obté en les darreres línies del codi és:

```
Example sequence
-----------------------
DNA Sequence #1:
 CCGAGGGCTA ... CGCGGACACC
```

```
One hot encoding of Sequence #1:
 [[0. 0. 0. ... 1. 0. 0.]
 [1. 1. 0. ... 0. 1. 1.]
 [0. 0. 1. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```

Un cop transformada la base de dades, cal carregar les etiquetes de la variable resposta a partir de instruccions del codi 3 de l'annex 1. En aquest cas, les etiquetes estan formades per un "1" que indica que la proteïna s'uneix a la seqüència mentre que un "0" significa que no.

```
Labels:
 [['0' '0' '0' ... '0' '1' '1']]
One-hot encoded labels:
 [[1. 1. 1. ... 1. 0. 0.]
 [0. 0. 0. ... 0. 1. 1.]]
```

L'estructura que segueix la variable resposta és la mateixa que les dades: cada valor es transforma en un vector de 2 dimensions, que si el vector és [1,0], el valor és "0" i si és [0,1] és valor és un "1".

Seguidament, es divideixen les dades en diferents grups per procedir a l'entrenament del model.

Modelatge i entrenament

Per començar, es tria una xarxa neuronal convolucional d'una dimensió. Com ja s'ha estudiat, les CNN aprenen a reconèixer patrons i característiques de les dades; amb les dades disponibles, es voldran reconèixer aquells *motifs* dins les seqüències d'ADN que permetin la unió de les proteïnes.

Per la construcció del model cal establir quines capes s'utilitzaran i especificar la seva dimensió:

- *Con1D*: Capa convolucional.
- *MaxPooling1D*: Capa que després del procés de convolució duu a terme el *pooling* per disminuir la mida de la base de dades.
- *Flatten*: Aquesta capa "aplana" les sortides obtingudes a la capa del *pooling*, combinant els resultats de la convolució i del procés del *pooling*.
- *Dense*: En el model també es considerarà una capa *dense* en la qual es comprimirà la informació de la capa *flatten* i aquesta es passarà a una nova capa *dense* per poder predir els valors de 0 o 1.

Totes aquestes capes s'ajuntaran en un model de tipus seqüencial (*sequential*), que es defineix com una pila lineal de capes.

Codi 2: Construcció del model

```
# Es carreguen les funcions que es necessitaran per fer el model
#de deep learning
from tensorflow.keras.layers import Conv1D, Dense, MaxPooling1D,
Flatten

# Es carrega el tipus de model que es durà a terme.
from tensorflow.keras.models import Sequential

# A contiuació es construeix el model a partir d'afegir (model.a
dd) capes al model sequential.
model = Sequential()
model.add(Conv1D(filters=32, kernel_size=12,
                 input_shape=(train_features.shape[1], 4)))
model.add(MaxPooling1D(pool_size=4))
model.add(Flatten())
model.add(Dense(16, activation='relu'))
model.add(Dense(2, activation='softmax'))
```

A continuació s'avaluarà cada un del paràmetres que es defineixen en les capes:

1. *Conv1D(filters=32, kernel_size=12, input_shape=(train_features.shape[1], 4))*: El paràmetre de filtres recull el nombre de filtres de sortida, és a dir, la dimensionalitat de l'espai de sortida. *Kernel_size* és un nombre que especifica la longitud de la finestra de convolució. El paràmetre de *input_shape* es necessita en les primeres capes del models seqüencials per definir la forma de les dades d'entrada. Aquest paràmetre estableix una capa de rebuda de les dades en la quals s'indica que és el primer valor de l'objecte *train_features.shape* i es determinen 4 neurones en aquesta capa.
2. *MaxPooling1D(pool_size=4)*: Es defineix la longitud que tindrà la finestra per dur a terme el procés de *pooling*.
3. *Flatten()*: En aquest cas, no cal especificar cap paràmetre que influeixi en el funcionament del model.
4. *Dense(16, activation='relu')*: Es defineix una capa *dense* en la qual s'hi aplicarà la funció d'activació *relu, .*
5. *Dense(2, activation='softmax')*: Finalment, per obtenir una sortida interpretable del model s'aplica una nova capa de tipus *dense* amb la qual hi treballarà el tipus de funció d'activació *softmax*, per la classificació dels dos valors.

En total es calcularan 6226 paràmetres.

Es pot observar que les capes convolucionals i de *MaxPooling* estan definides en una dimensió. Hi ha capes d'aquest tipus possibles en 2 i 3 dimensions, i els seus paràmetres seran similars. Cada una d'aquest tipus de capes s'utilitzaran per dades que tinguin les respectives dimensions.

Un cop construït el model, es compila i amb la comanda *model.summary()* es pot veure un petit resum del model:

Codi 3: Instruccions finals del model

```
# Es compila el model a partir de la funció model.compile
model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['binary_accuracy'])

# Mostra els elements del model
model.summary()
```

En la compilació del model s'ha de definir la funció de pèrdua que s'utilitzarà per l'optimització del model, el tipus d'optimitzador i les mètriques es que es volen obtenir per la seva avaluació. S'utilitza com a funció de pèrdua la *binary_crossentropy,* que mesura el rendiment d'un model de classificació, en aquest cas binària, on la sortida és una probabilitat. L'optimitzador que s'utilitza en aquest cas és el *adam,* algoritme d'optimització semblant al de de descens de gradient estocàstic, però amb alguns avantatges que aquest últim no té: manté una taxa d'aprenentatge per cada paràmetre que millora el rendiment en problemes amb gradients separats i també funciona amb problemes en línia (*online*) i no estacionaris. Finalment, es vol veure l'*accuracy* indicant-ho a la mètriques.

El model ja està llest per ser entrenat. Mitjançant la funció *model.fit* s'entrenarà el model, en el qual cal indicar-hi les dades d'entrenament amb les respectives etiquetes de la variable resposta, definir el nombre d'iteracions que es duran a terme (*epochs*), el paràmetre *verbose* és el que permet veure en pantalla l'evolució dels resultats durant les iteracions i en quin nivell (0 no es mostrarà res, 1 es mostrarà una barra de procediment i 2 s'indicarà la iteració que s'està executant). Finalment, amb el paràmetre de *validation_split* és defineix quin tant per cent de les dades d'entrenament correspon a les dades de validació, en aquest cas, un 25%.

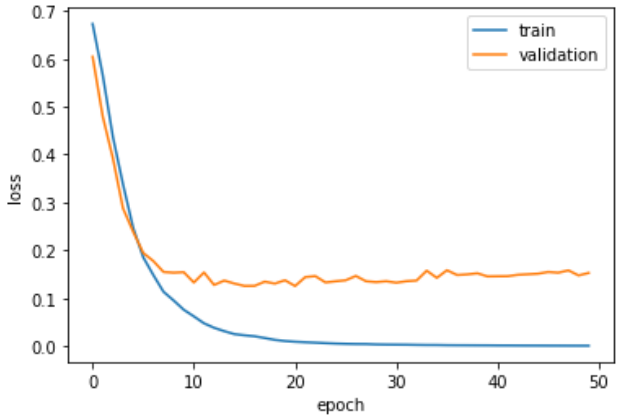
Codi 4: Ajust del model

```
history = model.fit(train_features, train_labels,
                    epochs=50, verbose=0, validation_split=0.25)
```

Avaluació

Un cop entrenat el model, es dibuixarà un gràfic on es mostra com disminueix la funció de pèrdua al llarg de l'entrenament del model, tant amb les dades d'entrenament com amb les de validació.

Gràfic 11: Evolució de l'ajust del model



En l'última iteració s'obté un *accuracy* de 0.9653 amb les dades d'entrenament i la pèrdua arriba a valors molt petits, com es pot observar en el gràfic 1.

Predicció

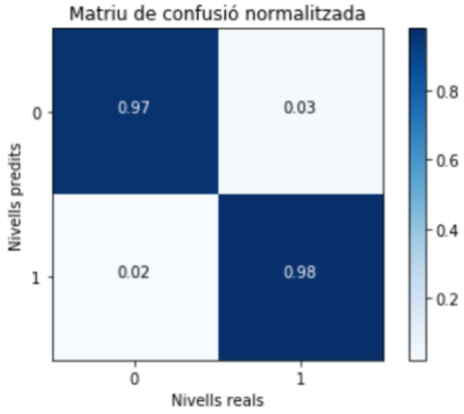
Tot i haver obtingut una precisió molt bona, per tal de veure si un model és bo per predir nous resultats és avaluar-lo sobre unes dades noves, és a dir, el conjunt de dades de test que s'havien guardat.

Codi 5: Avaluació del model

```
from sklearn.metrics import confusion_matrix
import itertools

predicted_labels = model.predict(np.stack(test_features))
cm = confusion_matrix(np.argmax(test_labels, axis=1),
                      np.argmax(predicted_labels, axis=1))
```

Amb la funció *model.predict* es poden predir els valor de les dades de test per posteriorment avaluar el seus resultats amb una matriu de confusió com la que es pot observar a continuació:

Gràfic 12: Matriu de confusió del model inicial

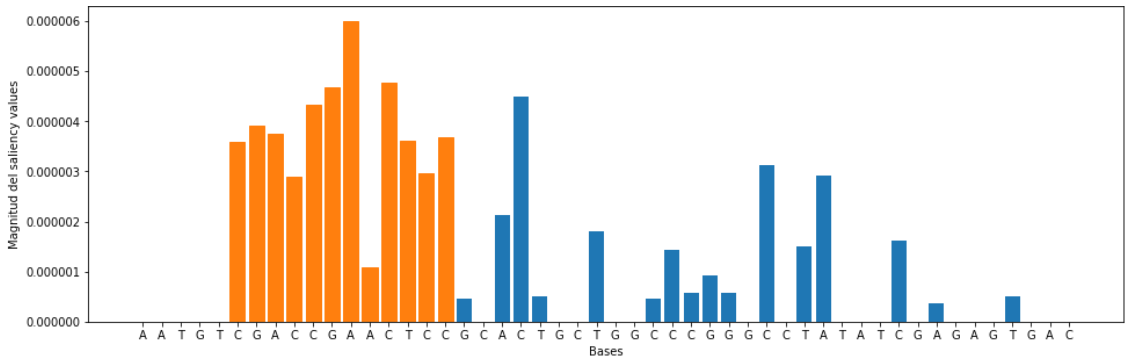


Aquesta matriu mostra que un 97% dels valors que eren 0 han estat ben predits i un 98% en el cas dels valors 1. Només un 2% s'ha equivocat quan havia de predir un 0 i un 3% quan havia de ser un 1.

Interpretació

Una vegada obtinguts els resultats del model i veure que aquest prediu bé cal saber interpretar què diuen els resultats. En molts casos, la interpretació dels paràmetres d'un model de *deep learning* és molt complicada, i tal i com diuen els autors de l'article, la aprendre a interpretar els paràmetres és un dels aspectes pendents en aquesta disciplina. En tot cas, per poder interpretar els resultats obtinguts en aquest problema, es dibuixa un *saliency map,* que dibuixa el gradient de la predicció respecte cada base nucleònica. En altres paraules, aquest tipus de gràfics mostren com canvia el valor de la resposta respecte petits canvis en la seqüència de nucleòtids d'entrada: els valors positius dels gradients diuen que un petit canvi en aquell nucleòtid canviarà el valor de sortida, per tant, la visualització d'aquesta gradients per una seqüència d'entrada ha de proporcionar algunes pistes sobre quins nucleòtids permeten la unió de la proteïna.

Gràfic 13: Interpretació de resultats (Saliency map)



La seqüència de bases que obté valors més alts és CGACCGAACTCC. És a dir, en aquesta seqüència de bases formen un punt d'unió de proteïnes.

6.1.3. Modificacions

A partir d'aquest exercici extret de l'article, es poden canviar alguns paràmetres per tal de poder veure quins canvis milloren el resultat de la predicció.

Un dels primers canvis és la modificació de de la funció d'activació de la segona capa *dense* per una *sigmoid*. En la resolució original s'utilitzava una *softmax* centrada en els problemes de més d'una categoria, en aquest cas, com que és un problema amb resposta binària, la nova funció també és adequada i pot significar alguns canvis. A més a més, durant el treball s'ha estudiat l'optimitzador *stochastic gradient descent,* pel que ara es vol provar com funciona aquest optimitzador i observar les diferències en el resultat respecte l'*adam*.

A partir d'aquí, es poden canviar alguns valors de les funcions com el nombre d'*epochs* que es duen a terme, que en el nous cas, s'especificarà en 30.

Amb els canvis duts a terme s'obté la següent matriu de confusió:

```
Matriu de confusió:
[[207  52]
[ 39 202]]
```

Tal i com es pot observar, els resultats de la predicció de la xarxa que s'ha executat són més dolents. Com l'optimitzador que s'aplica no presentava millores respecte original, l'*adam*, es prova d'augmentar el nombre d'iteracions a realitzar:

```
Matriu de confusió:
 [[243  16]
 [  6 235]]
```

Així, augmentant el nombre d'iteracions a 50 el resultat millora notablement. Tot i això, com que s'obtenien millors resultats amb l'optimitzador *adam* es segueix utilitzant aquest.

De nou amb l'*adam* com a optimitzador, es prova com es combina amb una funció d'activació *sigmoid* en la darrera capa de *dense*:

```
Matriu de confusió:
 [[250   9]
 [  6 235]]
```

S'obtenen resultats similars en els que es s'obtenien amb el problema original pel que fa la matriu de confusió, però l'*accuracy* obtingut en aquest cas és de 0.9787, pel que es pot considerar que hi ha millora respecte el resultat original.

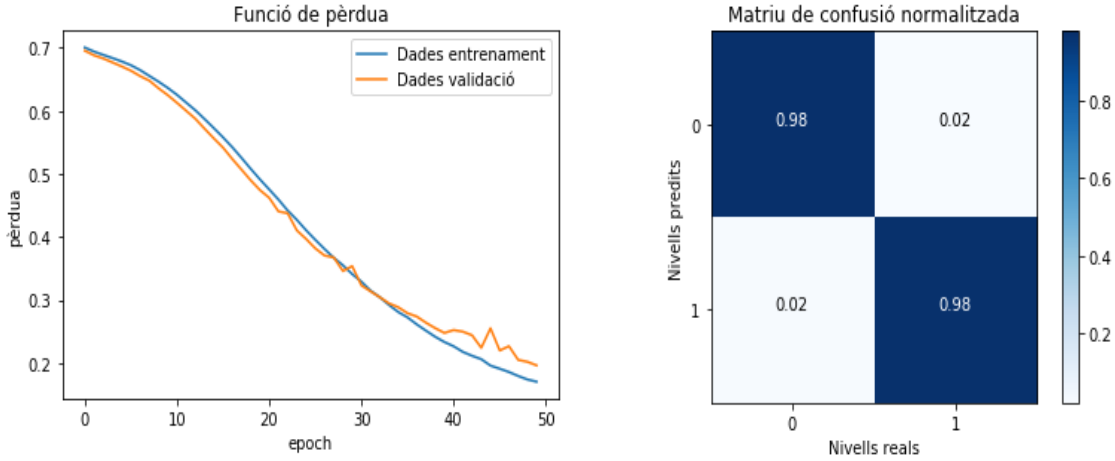
Durant els canvis anteriors, s'han obtingut variacions en els resultats i en alguns casos millores seguint una mateixa estructura del model i de les capes (hiperparàmetres), però ara es considera canviar la xarxa convolucional a una xarxa recurrent per veure com funciona aquesta en seqüències genòmiques. A continuació es mostra el codi amb les instruccions principals.

Codi 6: Noves modificacions

```
from tensorflow.keras.layers import LSTM, Dense

model = Sequential()
model.add(LSTM(32, input_shape=(train_features.shape[1], 4)))
model.add(Dense(16, activation='relu'))
model.add(Dense(2,activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='adam', metr
ics=['binary_accuracy'])

history = model.fit(train_features, train_labels,
                    epochs=50, verbose=1, validation_split=0.25)

predicted_labels = model.predict(np.stack(test_features))
```

Tal i com es pot veure, és una xarxa neuronal recurrent molt senzilla, la qual es defineix amb 32 neurones, i es mantenen les altres dues capes *dense*. Els resultats són els següents:

Gràfic 14: Avaluació del nou model

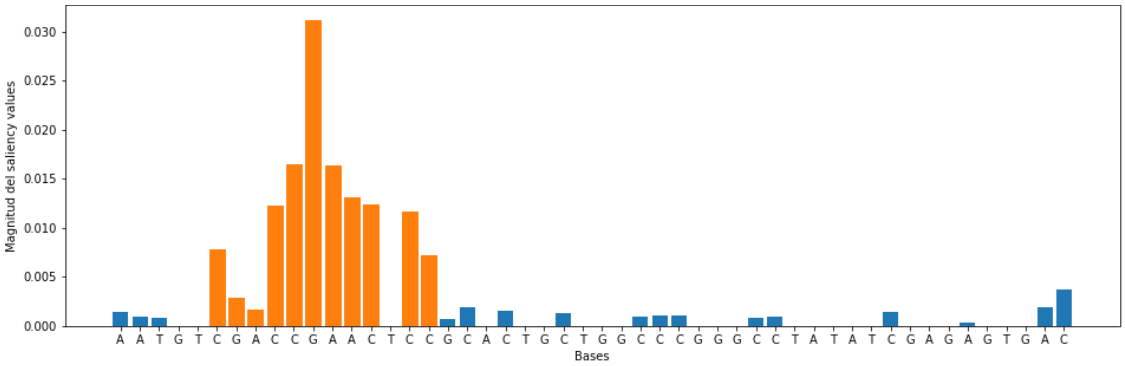


Cal destacar el comportament de la funció de pèrdua, com aquesta té un comportament menys marcat que el que tenia l'anterior i la seva millora és de forma més progressiva, tot i això, s'arriba a valors similars del que s'obtenien abans. Tot i arribar a valors similars, aquest progrés regular fa que no sigui fins la última iteració que s'obtinguin valors similars; en una xarxa CNN els resultats de la funció de pèrdua eren similars a la iteració 30 que a la 50. En general, es pot dir que l'aprenentatge és més lent.

Pel que fa la matriu normalitzada, es poden observar valors lleugerament millors que en la xarxa original, al igual que l'*accuracy* obtingut.

Pel que fa a la interpretació del problema amb una CNN, es torna a dibuixar el *saliency map* i el resultat és el següent:

Gràfic 15: Saliency map per la predicció del nou model



Les bases resultants es destaquen més que les altres, tot i que hi ha una d'aquestes que no obté valor.

Conclusió

Les conclusions que s'han dut a terme al llarg del problema es recullen en la següent taula:

Taula 2: Resum de resultats de les modificacions aplicades

Descripció	Matriu de confusió	*Accuracy*
Problema original	$\begin{bmatrix} 250 & 9 \\ 6 & 235 \end{bmatrix}$	0,9653
SGD Espoch: 30 Sigmoid	$\begin{bmatrix} 182 & 77 \\ 25 & 216 \end{bmatrix}$	0,7582
SGD Espoch: 50 Sigmoid	$\begin{bmatrix} 242 & 17 \\ 15 & 226 \end{bmatrix}$	0,9413
Adam Epoch: 50 Sigmoid	$\begin{bmatrix} 250 & 9 \\ 6 & 235 \end{bmatrix}$	0,9787
RNN	$\begin{bmatrix} 250 & 5 \\ 6 & 235 \end{bmatrix}$	0,9680

Tot i que s'han anat provant diferents formes de resolució del mateix problemes, s'aconsegueix en tots els casos bons resultats de precisió i predicció.

6.2. Aplicació 2: Predicció de l'abecedari ASL

6.2.1. Descripció

L'aplicació que es realitzarà a continuació és una demostració que l'alta tecnologia com poden ser les xarxes neuronals o el *deep learning* també es poden aplicar en àmbits que toquin ben de prop a la societat, en aquest cas, el col·lectiu de persones sordes americanes, i, que a més a més, poden presentar millores significatives en la comunicació d'aquest grup de persones, millorant el seu nivell de vida. Així doncs, el *deep learning* també es pot utilitzar per dur a terme una tasca social.

Per contextualitzar, la llengua de signes és el llenguatge a través de gestos pel que es comuniquen principalment les persones amb menor capacitat auditiva i/o amb dificultat a l'hora de parlar. També l'utilitzen les persones que pel motiu que sigui no tenen domini sobre la llengua parlada. A partir de la realització de gestos, aquest col·lectiu de persones representen paraules, situacions o lletres per comunicar-se, acompanyat també de l'expressió facial (rostre, ulls i boca) i sorolls que acompanyen al que diuen.

L'abecedari ASL és la llengua de signes americana, utilitzada als Estats Units, Canadà i algunes parts de Mèxic. Al igual que existeixen diferents llengües parlades, també existeixen diferents llengües signades, i, en aquest cas, la llengua americana signada manté forces diferències a la llengua que es parla als països d'Anglaterra.

Il·lustració 7: Representació d'ASL

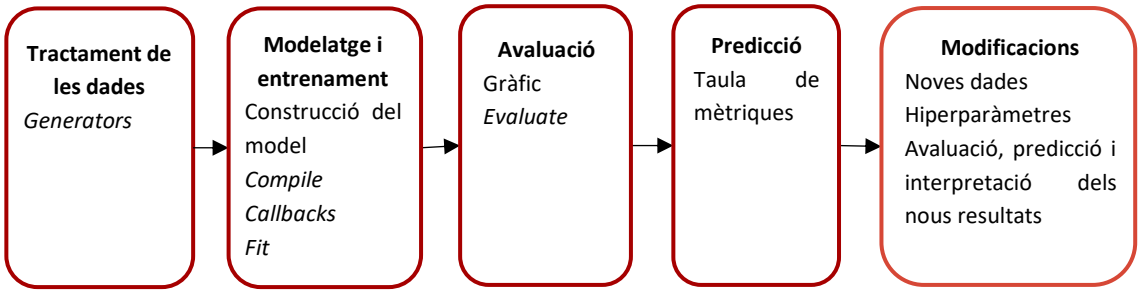
La base de dades que es presenta a continuació conté fotos de l'abecedari de la llengua de signes americana juntament amb el signe de borrar, d'espaiar i "res" (no es mostra cap signe a la fotografia). A la dreta del text es pot observar una imatge que conté el signe per cada una de les lletres de l'abecedari americà.

Les dades d'entrenament tenen 87.000 imatges de 200x200 píxels cada una. Existeixen 29 classes entre les lletres i el 3 signes de borrar, espai i "res", pel que corresponen 3.000 imatges a cada caràcter.

6.2.2. Resolució

Els passos que es seguiran per resoldre el següent problema són el següents:

Gràfic 16: Esquema de la resolució



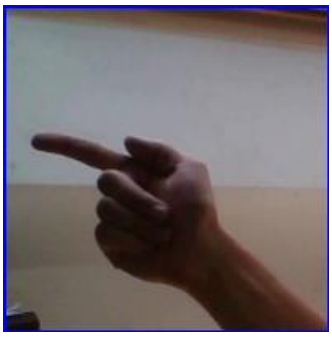
Per començar, cal carregar els paquets i directoris necessaris per l'execució del programa, que en aquest cas, es carrega cap de nou respecte el problema 1. Pel que fa les dades, aquestes estan emmagatzemades en carpetes, on cada carpeta rep el nom de la lletra que s'hi guarda. Com que les dades seran carregades a partir dels directoris on estan guardades, les funcions per entrenar el model hauran de ser de tipus *generator*.

Tractament de les dades

Tal i com s'ha anat comentat al llarg del treball i com s'ha pogut veure en l'exemple anterior, el tractament de dades abans d'introduir-les al model és molt important pel funcionament i l'eficiència d'aquest. Primer de tot, s'han de llegir les dades, com que estan en format JPEG, cal codificar-ho a formar de RGB, convertir les dades en tensors en punt flotant i finalment reescalar els valors dels píxels, que es troben entre 0 i 255. Una mostra de les dades originals són les següents:

Lletra I

Lletra V



Lletra G

Per dur a terme tot aquest procés, Keras incorpora en les seves funcions diferents eines per fer aquests passos automàticament a partir de generadors. Els generadors són funcions que permeten obtenir resultats a poc a poc, és a dir, permet obtenir dades en temps d'execució i, en aquest cas, a partir de les carpetes. En Keras, aquestes eines es troben en *ImageDataGenerator* que duu a terme les funcions de generadors en lots de dades.

Codi 7: Tractament de les imatges

```
train_data_gen=ImageDataGenerator(rescale=1./255,
                                  validation_split=0.2)

test_data_gen=ImageDataGenerator(rescale=1/.255)

train_generator=train_data_gen.flow_from_directory(
                train_dir,
                target_size=(64,64),
                batch_size=32,
                class_mode='categorical',
                subset='training')

validation_generator=train_data_gen.flow_from_directory(
                train_dir,
                target_size=(64,64),
                batch_size=32,
                class_mode='categorical',
                subset='validation',
                shuffle=False)

test_generator=test_data_gen.flow_from_directory(
                test_dir,
                target_size=(64,64),
                batch_size=1,
                class_mode='categorical',
                shuffle=False)
```

Per començar, es re-escalen les imatges de les dades d'entrenament i les de test per obtenir valors que es trobin entre el 0 i el 1. En el cas de les dades d'entrenament, també s'especifica un paràmetre per determinar quin nombre de dades formaran part del conjunt de validació.

A continuació a partir de *flow_from_directory* s'agafen les dades del directori, s'especifica quina mida tindran aquestes i de quina classe formen les dades. Com a sortida, es mostra el número de dades dins de cada conjunt: especificant un 20% de dades de validació, hi hauran 69600 imatges com a dades d'entrenament, 17400 imatges pel conjunt de validació i per avaluar la predicció del model, es reserva una imatge de cada classe.

Modelatge i entrenament

Amb les dades en un estat que poden ser tractades pel model, es pot passar a construir el model que s'utilitzarà, en aquest cas:

Codi 8: Definició del model

```
model=Sequential()
model.add(Conv2D(32,kernel_size=(3,3),input_shape=(64,64,3),acti
vation='relu',padding='same'))
model.add(Conv2D(32,kernel_size=(3,3),activation='relu',padding=
'same'))
model.add(MaxPooling2D(pool_size=(3,3)))
model.add(Conv2D(64,kernel_size=(3,3),activation='relu',padding=
'same'))
model.add(Conv2D(64,kernel_size=(3,3),activation='relu',padding=
'same'))
model.add(MaxPooling2D(pool_size=(3,3)))
model.add(Conv2D(128,kernel_size=(3,3),activation='relu',padding
='same'))
model.add(Conv2D(128,kernel_size=(3,3),activation='relu',padding
='same'))
model.add(MaxPooling2D(pool_size=(3,3)))
model.add(Flatten())
model.add(Dropout(0.45))
model.add(Dense(512,activation='relu'))
model.add(Dense(29,activation='softmax'))

model.summary()
model.compile(loss='categorical_crossentropy',metrics=['accuracy
'],optimizer='adam')
```

En el model que es defineix en considera una capa convolucional de dues dimensions; en aquest problema les dades tenen dues dimensions a diferència del problema anterior. Aquesta primera capa tindrà 32 resultats de sortida, s'especifica la matriu de filtre (*kernel_size=(3,3)*) i el tipus d'activació de la xarxa. En el problema anterior, les xarxes de convolució no se li especificava cap activació, ja que en la funció aquesta pot prendre el valor "*None*". El paràmetre *padding=same* significa que la mida de les dades d'entrada ha de ser la mateixa que les dades de sortida. Es va afegint més capes de convolució augmentat el seu nombre de dades de sortida i combinant amb capes de *pooling,* especificant una mida de finestra de 3x3 (*pool_size=(3,3)*). Per acabar, s'arriba a la capa *flatten* i tot seguit una capa anomenada *dropout* que ajuda a prevenir l'*overfitting.* Finalment, amb les capes *dense* es busca obtenir valors més interpretables del model aconseguint un percentatge per cada nivell de sortida.

En aquest cas, el model es compilarà amb la funció de pèrdua de *categorical_crossentropy,* es calcularà la mètrica de l'*accuracy* i s'utilitzarà l'optimitzador *adam.*

Un resum del model és el següent:

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 64, 64, 32)        896
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 64, 64, 32)        9248
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 21, 21, 32)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 21, 21, 64)        18496
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 21, 21, 64)        36928
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 7, 7, 128)         73856
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 7, 7, 128)         147584
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 2, 2, 128)         0
_________________________________________________________________
flatten_2 (Flatten)          (None, 512)               0
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_3 (Dense)              (None, 512)               262656
_________________________________________________________________
dense_4 (Dense)              (None, 29)                14877
=================================================================
Total params: 564,541
Trainable params: 564,541
Non-trainable params: 0
_________________________________________________________________
```

S’estimaran en total 564541 paràmetres. Amb el nombre de dades disponibles i el número de paràmetres a estimar, per cada un d’ells s’utilitzarà un nombre de 0,12 imatges.

El següent pas correspon a introduir les funcions per l’execució del model, però abans, hi ha algunes pràctiques avançades dins del *deep learning* que milloren el rendiment del model i que permeten solucionar alguns problemes. Una d’aquestes eines avançades són les *callback* de Keras i s’utilitzen, per exemple, per ajustar automàticament valors de certs paràmetres durant el model o per interrompre el model quan la funció de pèrdua no segueix disminuint. En aquest cas s’utilitzen dues, la primera s’anomena *ReduceLROnPlateau*, utilitzada per reduir la taxa d’aprenentatge (*learning rate*, hiperparàmetre que controla quant s’estan ajustant els pesos de la xarxa respecte la pèrdua) quan la pèrdua ja deixat de millorar, d’aquesta manera s’aconsegueix sortir dels mínims locals durant l’entrenament. La segona és *EarlyStopping* que serveix per interrompre l’entrenament una vegada que la mètrica que s'està supervisant hagi deixat de millorar per a un nombre fix d'èpoques.

Codi 9: Definció de les callbacks

```
callbacks=[
    keras.callbacks.ReduceLROnPlateau(
        monitor='val_loss',
        factor=0.1,
        patience=10),
    keras.callbacks.EarlyStopping(
        monitor='acc',
        patience=4)
]
```

En aquest cas, per la primera *callback* s'especifica quin tipus de pèrdua haurà de controlar (*monitor='val_loss'*), es divideix per 10 quan l'objecte s'activa (*factor=10*) i finalment, l'objecte s'activa quan la pèrdua de validació ha deixat de millorar en 10 *epochs* (*patience=10*). Pel que fa la segona *callback* es supervisarà en aquest cas l'accuracy (*monitor='acc'*) i s'activarà quan aquesta mesura hagi deixat de millorar durant 4 *epochs* (*patience=4*).

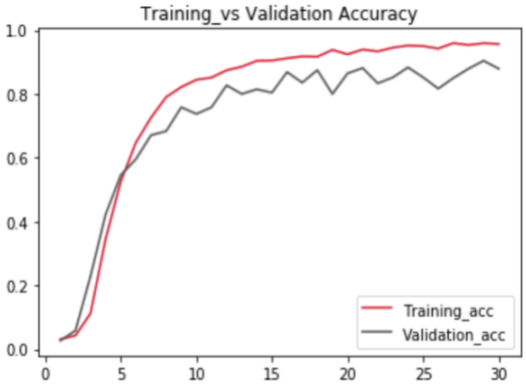
Hi ha establides algunes funcions de *callbacks*, però Keras també permet la construcció d'aquestes segons les necessitats dels problemes.

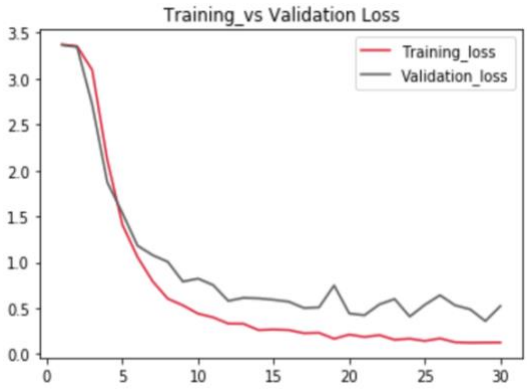
A continuació, es pot seguir amb el desenvolupament habitual de problemes: indicar les instruccions per executar el model. En aquest cas, s'utilitzarà la funció *fit_generator*, equivalent a la funció *fit* utilitzada en l'anterior problema per dades amb generadors. El primer que necessita aquesta funció és un generador de dades que proporcionarà els lots. Aquestes, com que es treballen durant el procés, el model necessita saber quantes mostres ha d'entrar abans d'acabar una *epochs* (*step_per_epochs=30*). També es pot especificar les dades de validació a través d'un generador (*validation_data=validation_generator*), però com en l'altre cas, també s'hauran d'especificar quantes mostres de dades s'hauran d'introduir en cada època (*validation_steps=10*). Finalment, s'introdueix la llista de *callbacks* que s'ha definit anteriorment.

Avaluació

Un cop executat el model hi ha diferents formes per veure els seus resultats. Es poden dibuixar dos gràfics veient el progrés de l'*accuracy* i la funció de pèrdua establerts al llarg de les *epochs.* En el primer gràfic es pot veure el progrés de l'*accuracy* al llarg de les iteracions: amb les dades d'entrenament s'aconsegueix un valor molt proper a 1 i amb les dades de validació arriba a un valor de 0.8792. Pel que fa la funció de pèrdua s'obtenen un valors de 0.12 per les dades d'entrenament i un 0.52 per les dades de validació.

Gràfic 17: Evolució del procés d'entrenament





A continuació s'avalua el model sobre les dades de validació. Per avaluar el model s'aplica la funció *evaluate_generator*:

Codi 10: Avaluació del model

```
evaluate=model.evaluate_generator(validation_generator,
                                  steps=544)
print("Loss:",evaluate[0])
print("Acc:",evaluate[1])
```

```
Loss: 0.4811159350924729
Acc: 0.8778199815837937
```

Avaluant el model sobre les dades de validació s'obté que la funció de pèrdua val 0.48 i un *accuracy* de 0.87. El valor varia del obtingut amb el procés perquè la funció *evaluate* genera una mitjana dels valors de la pèrdua i del *accuracy* en cada grup de dades que s'agafa. En aquest cas, cada un d'aquests valors es divideix per 544 com a resultat de dividir el nombre de mostres en les dades de validació entre el nombre de dades que s'agafen en cada grup.

Predicció

Pel que fa el nivell de predicció del model, en problemes que hi ha més de dues classes per predir, es pot construir la següent taula:

Taula 3: Resultats de predicció del model amb les dades de validació

Classification Report

	precision	recall	f1-score	support
A	0.81	0.96	0.88	600
B	1.00	0.74	0.85	600
C	0.93	1.00	0.96	600
D	0.98	0.99	0.99	600
Del	0.72	0.95	0.82	600
E	0.96	0.95	0.95	600
F	0.96	0.50	0.66	600
G	0.61	1.00	0.76	600
H	1.00	0.74	0.85	600
I	0.82	0.89	0.85	600
J	0.97	0.96	0.96	600
K	1.00	1.00	1.00	600
L	0.63	0.81	0.71	600
M	0.69	0.49	0.57	600
N	0.93	0.79	0.85	600
Nothing	0.98	0.95	0.97	600
O	0.97	1.00	0.99	600
P	0.93	0.95	0.94	600
K	0.65	0.82	0.73	600
R	0.98	0.76	0.85	600
S	0.91	0.96	0.94	600
Space	0.92	0.93	0.93	600
R	0.89	0.94	0.92	600
U	0.96	0.56	0.71	600
V	0.91	0.93	0.92	600
W	0.88	0.97	0.92	600
X	0.98	0.96	0.97	600
Y	0.95	0.96	0.96	600
Z	0.99	1.00	0.99	600
micro avg	0.88	0.88	0.88	17400
macro avg	0.89	0.88	0.88	17400
weighted avg	0.89	0.88	0.88	17400

En aquesta taula es pot observar per cada una de les 29 classes a estimar el seu valor de *precision, recall, f1-score* i el nombre de dades sobre el qual s'han calculat (en aquest cas, sobre les dades de validació). La mesura *f1-score* correspon a una mitjana ponderada entre el *precision* i *recall,* on el millor valor és 1 i 0 el seu pitjor.

El nombre de dades disponibles per cada categoria en les dades de test és d'una imatge, per tant, no hi ha les dades suficient per fer una bona avaluació del model. Si es prova de dur a terme les anteriors instruccions per les dades de test es podrà veure que els resultats són dolents.

6.2.3. Modificacions

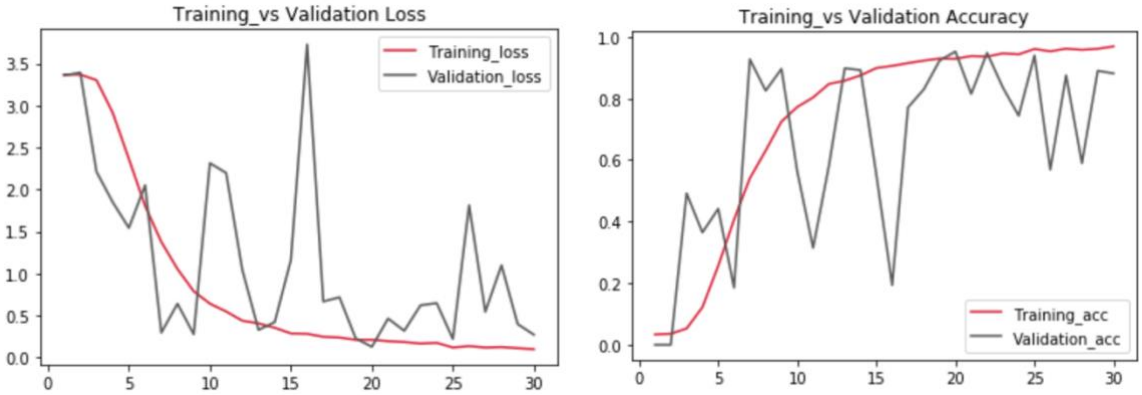
Davant d'aquesta situació, es considera la incorporació de noves imatges en les dades de test extretes de les dades d'entrenament: s'introdueixen 200 imatges per lletra en el grup test, més la que es disposava per defecte, es tindran doncs 201 dades. La distribució de les 87000 dades queda:

```
Found 64960 images belonging to 29 classes.
Found 16240 images belonging to 29 classes.
Found 5829 images belonging to 29 classes.
```

Això és una disminució de 4640 imatges en el grup d'entrenament i de 1160 en el grup de validació, pel que, utilitzant el mateix model, s'utilitzaran 0.11 imatges per cada paràmetre.

S'entrena el model amb els nous conjunt de dades; l'evolució de la pèrdua i de la precisió obtinguda al llarg de les iteracions és la següent:

Gràfic 18: Avaluació del nou model



Comparant els anteriors gràfics amb els gràfics obtinguts amb la resolució inicial, es pot veure clarament que les dades de validació tenen un comportament més irregular i tarden més en convergir: sembla ser que un nombre de 30 *epochs* no s'aconsegueixen estabilitzar les dades de validació. Caldria doncs, augmentar els valors dels paràmetres del model per que fa les *epochs* i el *steps_per_epoch* per poder fer un millor tractament de les dades disponibles.

A continuació, s'avalua el model amb les dades de validació i els resultats són els següents:

```
Loss: 0.5461474592727072
Acc: 0.8269088669950739
```

Els valors no són diferents els que s'havien obtingut amb el model amb més dades, però en l'altra cas els resultats eren millor que aquests últims. Tot i això, i tal i com s'ha fet anteriorment, es pot fer calcular la taula de *classification report* amb les dades de test.

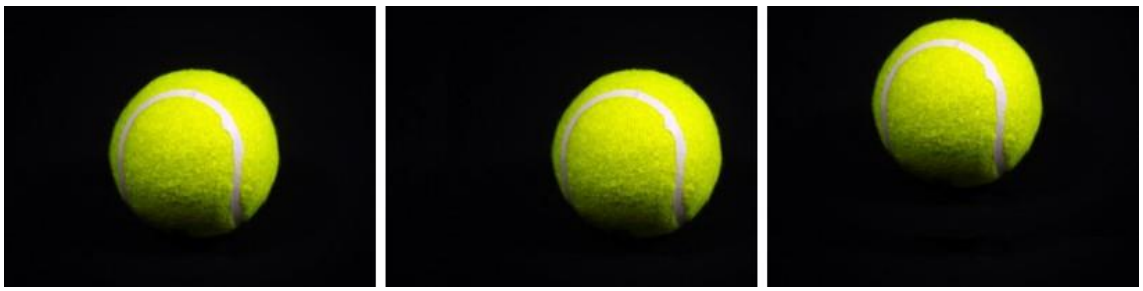
Taula 4: Resultats de la predicció del model amb les dades test

	precision	recall	f1-score	support
A	0.22	0.16	0.18	560
B	0.23	0.23	0.23	560
C	0.22	0.20	0.21	560
D	0.29	0.23	0.26	560
Del	0.16	0.24	0.19	560
E	0.20	0.23	0.21	560
F	0.16	0.15	0.15	560
G	0.27	0.33	0.29	560
H	0.21	0.15	0.18	560
I	0.29	0.27	0.28	560
J	0.22	0.23	0.22	560
K	0.22	0.23	0.23	560
L	0.11	0.11	0.11	560
M	0.37	0.35	0.36	560
N	0.19	0.20	0.19	560
Not	0.20	0.23	0.21	560
O	0.27	0.23	0.25	560
P	0.23	0.23	0.23	560
Q	0.11	0.17	0.13	560
R	0.24	0.14	0.18	560
S	0.15	0.20	0.17	560
Space	0.26	0.23	0.24	560
T	0.28	0.34	0.31	560
U	0.16	0.09	0.12	560
V	0.23	0.22	0.23	560
W	0.24	0.23	0.23	560
X	0.24	0.24	0.24	560
Y	0.22	0.23	0.23	560
Z	0.20	0.23	0.21	560
micro avg	0.22	0.22	0.22	16240
macro avg	0.22	0.22	0.22	16240
weighted avg	0.22	0.22	0.22	16240

Tot i que només es disposava d'una sola imatge per conjunt de dades test, reduir el nombre de dades d'entrenament no és una bona tècnica per tal d'avaluar les dades test, ja que, per una banda, s'obté un model més dolent, i per l'altra, els resultats de la taula anterior no milloren tot i augmentar el nombre de dades. Cal destacar que la quantitat de dades que s'ha utilitzat en les dades de test en el segon cas era inferior al 10% de les dades total, percentatge mínim recomanat per la definició d'aquest conjunt de dades. Si s'haguessin agafat més dades, l'aprenentatge del model hagués empitjorat.

Per problemes de mancança de dades es podria realitzar la tècnica de *data augmentation* amb la qual, a partir de les dades disponibles en la es creen noves dades com les originals introduint petites variacions. Aquesta tècnica s'utilitza en imatges i les modificacions que s'introdueixen són rotacions de les imatges o ampliar o reduir els elements d'aquesta, un exemple es representa en la següent imatge:

Il·lustració 8: Exemple de data augmentation



Com es pot veure en la imatge anterior, a partir de la primera fotografia de la pilota es poden crear dues més.

El codi juntament amb els seus resultat d'aquest segon model amb la nova distribució de les dades es troba a l'annex 3.

6.3. Aplicació 3: *Sentiment analysis*

6.3.1. Descripció

Des del reconeixement d'imatges fins les la predicció d'un valor, les xarxes neuronals poden resoldre gran varietat de problemes. D'entre totes les possibilitats, també destaca l'interès per veure el comportament de les persones a les xarxes socials, així com les reaccions davant d'un fet o la seva opinió. En aquest sentit, Twitter ha esdevingut una gran font d'informació per moltes empreses i institucions.

Existeixen diferents bases de dades disponibles on es mostra per diferents piulades si la reacció a un tema determinat és positiva o negativa. En aquest cas, la base de dades que es treballarà a continuació mostra diferents *tweets* reaccionant a la retransmissió del debat que hi va haver del passat 7 d'agost a Ohio per les eleccions generals del districte.

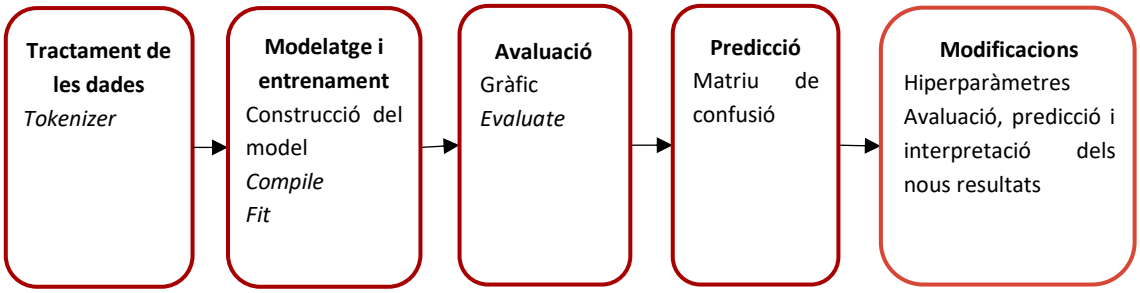
La base de dades original està disponible a la plataforma Figure Eight, però la que s'utilitzarà en el següent problema és una versió modificada de l'anterior, obtinguda a partir de la plataforma Kaggle. La base de dades conté 27742 piulades en les quals s'avalua la rellevància del *tweet*, de quin candidat està parlant, quin tema traca i quina és la reacció de la persona enfront del que parla.

L'objectiu de treballar aquesta base de dades és aplicar una xarxa neuronal recurrent de tipus LSTM, per tal de poder aprendre les funcions de Keras que s'han d'utilitzar per la definició d'aquest model com per el tractament de les dades. Aquest model de *deep learning* podrà predir si la reacció d'un *tweet* és positiva i negativa demostrant així la potència d'aquesta eina davant a la investigació del comportament humà en les xarxes socials i el posicionament envers temes d'interès econòmic o social.

6.3.2. Resolució

La resolució completa del problema, mostrant la sintaxi de Python i els resultats corresponents, es troba al Annex 3 del treball i a continuació es presenta un esquema de les parts que es seguiran:

Gràfic 19: Esquema de la resolució



Com en els anteriors problemes, per començar, s'han de carregar els paquets necessaris pel desenvolupament de la xarxa i carregar la base de dades. En aquest cas només es treballarà amb les variables referents al text de la piulada i la reacció d'aquesta. Pel que fa les reaccions, només es mantenen els nivells "*Positive*" i "*Negative*", amb les quals s'obtenen 4472 resultats per la primera categoria i 16986 per la segona.

Tractament de les dades

A continuació es passa a tractar la variable text amb les funcions que es mostren en el següent codi, però abans, un exemple del text cru és el següent:

"RT @ScottWalker: Didn't catch the full #GOPdebate last night. Here are some of Scott's best lines in 90 seconds. #Walker16 http://t.co/ZSfF..."

Codi 11: Tractament de les dades

```
# Conservar només aquells tweets que siguin text o paraules
data['text'] = data['text'].apply(lambda x: x.lower())
data['text'] = data['text'].apply((lambda x: re.sub('[^a-zA-z0-9
\s]','',x)))

for idx,row in data.iterrows():
    row[0] = row[0].replace('rt',' ')

# Nombre màxim de paraules per tweet i definició de seqüències
max_fatures = 2000
tokenizer = Tokenizer(num_words=max_fatures, split=' ')
tokenizer.fit_on_texts(data['text'].values)
X = tokenizer.texts_to_sequences(data['text'].values)
X = pad_sequences(X)
```

El primer tractament que s'aplica sobre la variable "text" és la funció *.apply* que aplica una funció determinada sobre les dades. En aquest cas, mitjançant el recurs de la lambda (s'utilitza per cridar funcions sense un nom concret) s'apliquen les funcions *.lower* i *re.sub*. La primera funció transforma el conjunt que s'especifica, en aquest cas amb *x*, en lletres minúscules i la següent funció s'utilitza per seleccionar només els valors alfanumèrics. Amb aquestes dues transformacions aplicades a la variable *tex*t, aquesta resulta (seguin el mateix exemple anterior):

"rt scottwalker didnt catch the full gopdebate last night here are some of scotts best lines in 90 seconds walker16 httptcozsff"

A continuació, s'eliminen els caràcters inicials referents a *retweet* (rt) a partir del procés iteratiu que hi ha indicat en el codi. Finalment s'utilitza la funció *Tokenizer*, una API de Keras que permet dividir el text en paraules o en textos secundaris, de forma que el model podrà crear relacions fàcilment entre el text i les etiquetes. En aquest cas, es defineix una funció *tokenizer* en la qual es permetrà un màxim de 2000 paraules separades per un espai. Seguint les línies del codi, la funció *fit_on_text* crea un diccionari intern on cada paraula prendrà un valor enter segons l'aparició d'aquesta. A

continuació, junt amb la funció *tokenizer* s'aplica *text_to_sequence* amb la qual es converteix cada text en una seqüència d'enters corresponents al índex que tenen en el diccionari definit. En l'exemple anterior (sense tenir en compte "*rt*"), la paraula "*didn't*" correspon al índex 252:

[16, 284, 252, 5, 821, 102, 167, 26, 136, 6, 1, 173, 12, 2, 233, 724, 17]

Aquests resultats es guarden en una matriu X en que se li aplica la funció *pad_sequence* en el qual passa cada un dels enters a una matriu, on cada fila correspondrà a una entrada de la variable:

[0 0 0 0 0 0 0 0 0 0 0 16 284 252 5 821 102 167 26 136 6 1 173 12 2 233 724 17]

D'aquestes dades que s'han tractat, cal dividir-les entre el conjunt d'entrenament i el conjunt de test, aplicant la funció *train_test_split* amb la que s'especifiquen les dades i el percentatge de dades que formaran les dades test, en aquest cas un 33%: Hi hauran 7188 entrades per les dades d'entrenament i 3541 per les dades de test.

Modelatge i entrenament del model

A continuació es defineix el model, tal i com s'ha fet en els anteriors problemes:

Codi 12: Definició del model

```
embed_dim = 128
lstm_out = 196

model = Sequential()
model.add(Embedding(input_dim=max_fatures,
                    output_dim=embed_dim,
                    input_length = X.shape[1]))
model.add(SpatialDropout1D(0.4))
model.add(LSTM(units=lstm_out, dropout=0.2, recurrent_dropout=0.
2))
model.add(Dense(2,activation='softmax'))

model.compile(loss = 'categorical_crossentropy',
              optimizer='adam',
              metrics = ['accuracy'])
print(model.summary())
```

S'observen algunes capes que no s'havien utilitzat fins ara. Com que s'aplica una xarxa neuronal recurrent es necessiten un conjunt de capes concretes per tractar les dades i poder ajustar el model. La primera d'elles, és la capa *Embedding,* necessària com a primera capa en una RNN per convertir els valors d'entrada amb tipus *Dense.* En aquesta funció s'hi indica el nombre de paraules que es tractaran, especificat en *max_features*, la mida de l'espai vectorial de sortida on es guardaran les paraules amb *output_dim* i finalment el nombre de llargada de les seqüències, en aquest cas, 28.

A continuació s'utilitza la capa *SpatialDropout1D* que funciona com un tipus de capa *Dropout,* però tractant en vectors d'una dimensió en comptes d'individus, seguida de la capa LSTM, en el qual s'especifica la mida que prendran els resultats de sortida, la taxa de del *dropout* de les dades d'entrada (*drop-out=0.2*) i de les dades mentre es treballin les transformacions lineals (*recurrent_dropout=0.2*). Per acabar, s'incorpora una capa *Dense* amb una funció d'activació *softmax.*

Amb aquest model s'hauran de calcular 511194 paràmetres; amb una base de dades de 27742 entrades, cada un dels paràmetres disposarà d'un 5% de les dades en total.

En resum, el model és:

```
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 28, 128)           256000
_________________________________________________________________
spatial_dropout1d_1 (Spatial (None, 28, 128)           0
_________________________________________________________________
lstm_1 (LSTM)                (None, 196)               254800
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 394
=================================================================

Total params: 511,194
Trainable params: 511,194
Non-trainable params: 0
```

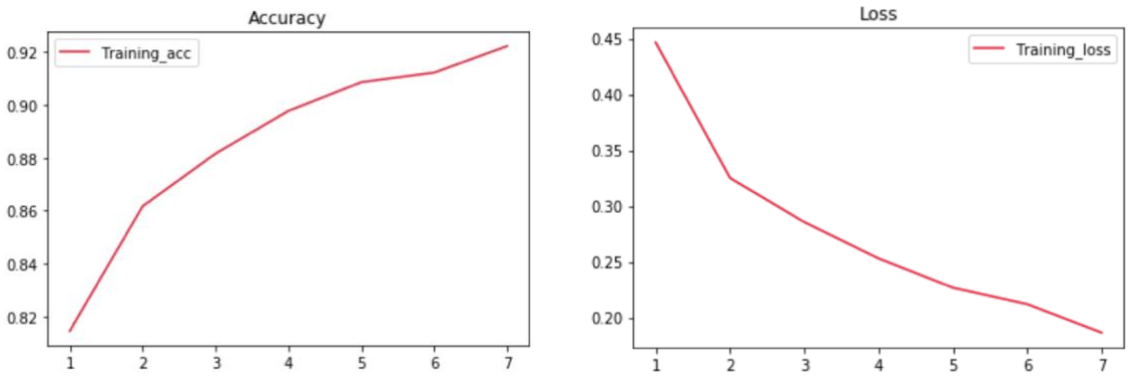
Finalment, es defineix que la funció de pèrdua serà la *categorical_crossentropy*, amb la funció d'optimització *adam* i calculat l'*accuracy* com a mètrica i ja es pot executar el Codi 6 de l'annex amb el qual s'ajusta i s'entrena el model i s'obtenen els resultats següents:

```
Epoch 1/7
 - 15s - loss: 0.4465 - acc: 0.8146
Epoch 2/7
 - 18s - loss: 0.3251 - acc: 0.8617
Epoch 3/7
 - 14s - loss: 0.2856 - acc: 0.8816
Epoch 4/7
 - 14s - loss: 0.2530 - acc: 0.8977
Epoch 5/7
 - 14s - loss: 0.2268 - acc: 0.9086
Epoch 6/7
 - 14s - loss: 0.2120 - acc: 0.9122
Epoch 7/7
 - 14s - loss: 0.1865 - acc: 0.9222
```

Avaluació

Com es pot observar, es duen a terme 7 iteracions de les dades agafant-les amb lots in dependents de 32 dades. Amb aquests paràmetres, s'aconsegueix un *accuracy* del 92% i una pèrdua de 0.1865. Es poden veure l'evolució del model amb els següents gràfics:

Gràfic 20: Evolució de l'accuracy i de la funció de pèrdua del model

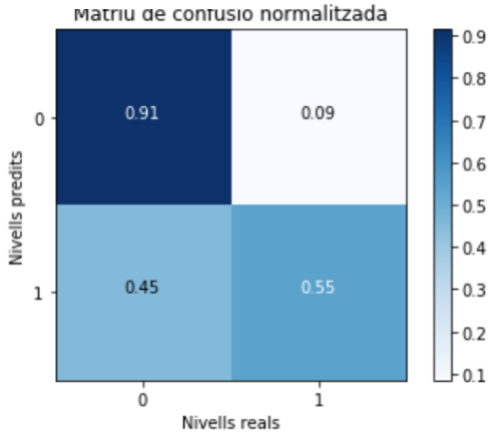


Per acabar amb el procés d'aprenentatge, s'avalua el model mitjançant la funció *evaluate,* especificant també el conjunt de dades de validació que es vol que s'utilitzi, amb el que s'obté un valor de l'*accuracy* del 82% i una pèrdua del 0.50.

Predicció

Finalment, s'aplica la funció *predict* i es dibuixa la matriu de confusió normalitzada:

Gràfic 21: Matriu de confusió normalitzada del model 1



Com es pot veure, tot i que s'obtenen valors bons pel que fa l'entrenament del model i l'avaluació sobre les dades de validació, quan es fa una matriu de confusió es pot veure com la predicció no és tant bona per una de les categories. Per la categoria "*Negative*" es prediuen correctament el 91% de les dades, però quan s'ha de predir la categoria "*Positive*" només un 55% de les dades es prediu bé.

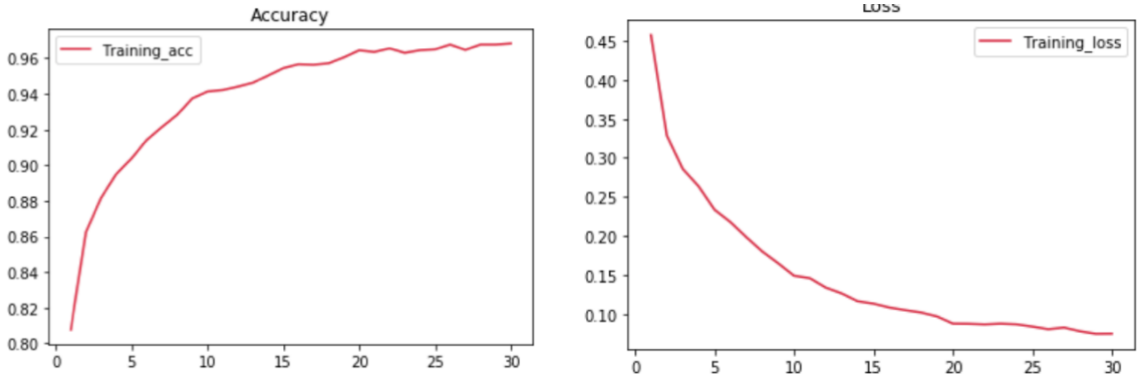
6.3.3. Modificacions

A partir d'aquests resultats, es duran a terme algunes modificacions al model anterior per intentar obtenir millors resultats.

En la definició del model, s'utilitzarà una funció d'activació *sigmoid* amb una funció de pèrdua *binary_crossentropy*, i un el *binary_accuracy*. A més a més, s'augmenta a 30 les *epochs* que es realitzen.

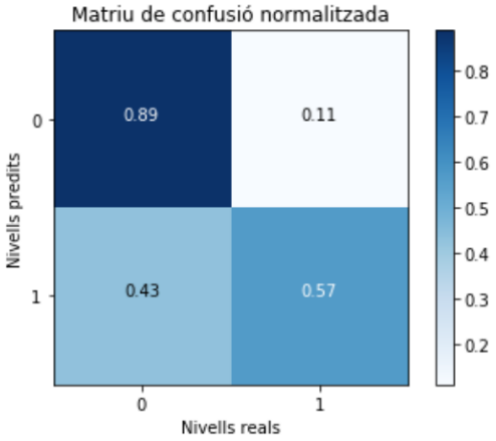
Els gràfics de l'aprenentatge que s'obtenen amb aquest nou model són els següents:

Gràfic 22: Evolució de l'acuraccy i la funció de pèrdua del la primera modificació



Com es pot observar els valors de l'*accuracy* i la funció de pèrdua milloren amb les dades d'entrenament. Com es comporta aquest nou model fent una predicció amb les dades de test? Configurant la matriu de confusió normalitzada es poden veure els resultats:

Gràfic 23: Matriu de confusió normalitzada de la primera modificació



Amb aquest nou model s'aconsegueix predir millor la classe "Positive", però el nombre de cops que es prediu bé la classe "*Negative*" es redueix.

Utilitzant l'anterior model, però incorporant les dades de validació en l'ajust d'aquest es pot veure en l'annex que no s'obté una gran millora respecte els resultats anterior

Així doncs, després d'haver provat diferents formes i paràmetres que poden ajudar a una millor predicció del model, un dels motius pels quals no s'arribi a aquest objectiu és la descomposició de les classes: és més freqüent "Negative" que "Positive". En problemes com aquests, es poden presentar diferents opcions com a solució:

1. Reduir la mida de les dades d'entrenament, reduint també el nombre de paràmetres a estimar.
2. Ampliar la base de dades fins aconseguir les mateixes freqüències en les dues categories.
3. Utilitzar una xarxa neuronal recurrent de tipus LSTM ja entrenada.

7. CONCLUSIONS

Tal i com es va comentar a la introducció, l'objectiu del present treball era poder aprendre i conèixer les metodologies del *deep learning*, així com estudiar les xarxes neuronals, el seu funcionament o l'estimació de paràmetres, entre d'altres.

Per una banda, amb la part teòrica del treball, s'han repassat els principis fonamentals de les xarxes neuronals, eina aplicada a diferents camps del *machine learning*: s'ha pogut veure els seus principis bàsics amb els seus fonaments, les possibles estructures de xarxes i com es desenvolupen estudiant el seu funcionament i finalment els problemes principals d'aquesta eina i com es poden corregir. Un cop establerts aquests punts, s'ha estudiat el *deep learning*, aprofundint en el seus mètodes d'iteració i estimació de paràmetres i estudiant els tipus d'estructura més útils per aquests problemes, les xarxes neuronals convolucionals i les xarxes neuronals recurrents.

Amb aquesta base s'ha pogut establir un coneixement teòric del tema del treball, però també a permès conèixer la biblioteca Keras i tenir un primer contacte amb la construcció de models de *deep learning*: la seva metodologia, el tractament de les dades, la modelització de les capes, establir els paràmetres, etc., però el més enriquidor ha sigut el fet de poder veure diferents aplicacions amb diferents dades. D'aquesta manera, s'ha ampliat el coneixement i l'experiència sobre el *deep learning*. S'han conegut funcions adaptades a cada tipus de dades, com s'havien de treballar aquestes, utilitzar les capes corresponents a cada model i buscar els paràmetres que puguin millorar els resultats del model.

Tot i això, s'ha pogut comprovar que l'experiència és un factor important en la modelització de xarxes neuronals en *deep learning*. Per exemple, l'experiència permet conèixer quines funcions es poder adaptar millor a les dades i quina combinació de capes cal utilitzar per poder aprendre sobre les dades determinades, de forma que l'estudi teòric de les xarxes neuronals juntament amb el deep learning i practicant sobre diferents dades i problemes, permet obtenir un treball més profund i a la vegada millors resultats.

En conclusió, aquest treball ha significat l'inici per seguir coneixent Keras i totes les seves funcionalitats, la base per la qual seguir estudiant diversos conceptes relacionats amb el *deep learning* i el *machine learning* en general, així com en la intel·ligència artificial.

8. BIBLIOGRAFIA

1. Nunes da Silva, Ivan, et al. *Artificial Neural Networks.* Suïssa : Springer, 2017. 78-3-319-43161-1.

2. Press, Academic. *Artificial Intelligence in the Age of Neural Networks and Brain Computing.* Regne unit : Elsevier, 2018. 978-0-12-815480.

3. GeeksforGeeks. Activation functions in Neural Networks. [En línea] [Consultat al: 23 de 3 de 2019.] https://www.geeksforgeeks.org/activation-functions-neural-networks.

4. *Some Asymptotic Results for Learning in Single Hidden-Layer Feedforward Network Models.* White, Halbert. 408, s.l. : American Statistical Association, 1989, Journal of the American Statistical Association, Vol. 84, págs. 1003-1013. 01621459.

5. Sarle, Warren S. FAQ. [En línea] 2012. [Consultat al: 17 de 3 de 2019.] ftp://ftp.sas.com/pub/neural/FAQ.html#A_data.

6. Sancho Caparrini, Fernando. Clasificación Supervisada y No Supervisada. [En línea] 29 de 12 de 2019. [Consultat el: 1 de 4 de 2019.] http://www.cs.us.es/~fsancho/?e=77.

7. #hub, I'mnovation. Aprendizaje reforzado: cuando las máquinas aprenden solas. [En línea] [Consultat al: 23 de 3 de 2019.] https://www.imnovation-hub.com/es/transformacion-digital/aprendizaje-reforzado-cuando-las-maquinas-aprenden-solas.

8. Bownlee, Jason. What is the Difference Between Test and Validation Datasets. [En línea] 2017. [Consultat al: 4 de 7 de 2019.] https://machinelearningmastery.com/difference-test-validation-datasets/.

9. Shah, Tarang. About Train, Validation and Test Sets in Machine Learning . [En línea] Towards Data Science, 2019. [Consultat al: 11 de 4 de 2019.] https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7.

10. Chollet, François. *Deep learning wih Python.* Estats Units d'Amèrica : Manning Publications Co, 2019.

11. learning, Introduction to deep.

12. MathWorks. Deep learning: tres cosas que es necesario saber. [En línea] [Consultat al: 7 de 3 de 2019.] https://es.mathworks.com/discovery/deep-learning.html.

13. Maruti. 8 problems that can be easily solved by machine learning. [En línea] [Consultat al: 28 de 3 de 2019.] https://www.marutitech.com/problems-solved-machine-learning.

14. Ian Goodfellow, et al. Deep learning. [En línea] MIT Press, 2016. [Consultat al: 5 de 3 de 2019.] http://www.deeplearningbook.org.

15. Nielsen, Micheal. Neurak Networks and Deep Learning. [En línea] Determination Press. [Consultat al: 6 de 3 de 2019.] http://neuralnetworksanddeeplearning.com.

16. Charniak, Eugene. *Introduction to deep learning.* Estats Units d'Amèrica : The Massachusetts Institute of Technology, 2018. 978-0-262-03951-2.

17. Torres, Jordi. Deep learning, Introducción pràctica con Keras. [En línea] 6 de 2018. [Consultat al: 19 de 4 de 2019.] https://torres.ai/deep-learning-inteligencia-artificial-keras/.

18. Jeong, Jiwon. The Most Intuitive and Easiest Guide for Recurrent Neural Network<gu. [En línea] Towards Data Science, 2017. [Consultat al: 22 de 4 de 2019.] https://towardsdatascience.com/the-most-intuitive-and-easiest-guide-for-recurrent-neural-network-87c29da73c7.

19. Olah, Christopher. Understanding LSTM Networks. [En línea] 2017. [Consultat al: 23 de 4 de 2019.] http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

20. Keras. Keras Documentation. [En línea] [Consultat al: 10 de 3 de 2019.] https://keras.io.

21. *A primer on deep learning in genòmics. A: Nature Genetics.* Zou, James. 1, 2019, Vol. 51, págs. 12-18. 1546-1718.

9. ANNEXES

9.1. Annex 1

Base de dades

```
# Codi 1

# Es carreguen els paquets necessàris, anteriorment instal·lats.
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import requests

# Es crea un objecte que contindrà la URL a partir de la
# qual s'obtindrà la base de dades
SEQUENCES_URL = 'https://raw.githubusercontent.com/abidlabs/deep-learn
ing-genomics-primer/master/sequences.txt'

# A partir del paquet "request" s'importa la base de dades i seguidame
nt s'eliminen les línies buides
sequences = requests.get(SEQUENCES_URL).text.split('\n')
sequences = list(filter(None, sequences))

# Amb el següent codi es mostren les cinc primeres seqüències
# de la base de dades
pd.DataFrame(sequences, index=np.arange(1, len(sequences)+1),
             columns=['Sequences']).head()
```

	Sequences
1	CCGAGGGCTATGGTTTGGAAGTTAGAACCCTGGGGCTTCTCGCGGA…
2	GAGTTTATATGGCGCGAGCCTAGTGGTTTTTGTACTTGTTTGTCGC…
3	GATCAGTAGGGAAACAAACAGAGGGCCCAGCCACATCTAGCAGGTA…
4	GTCCACGACCGAACTCCCACCTTGACCGCAGAGGTACCACCAGAGC…
5	GGCGACCGAACTCCAACTAGAACCTGCATAACTGGCCTGGGAGATA…

```
# Codi 2

from sklearn.preprocessing import LabelEncoder, OneHotEncoder

# Codifica una seqüència de bases com una seqüència de enters.
integer_encoder = LabelEncoder()
# converteix una matriu d'enters en una matriu escassa on
# cada fila correspon a un possible valor de cada funció.
one_hot_encoder = OneHotEncoder(categories='auto')
input_features = []

# Amb un for es va tranformant la base de dades
for sequence in sequences:
  integer_encoded = integer_encoder.fit_transform(list(sequence))
# Fit_tranform: Ajusta el codificador d'etiquetes i torna
# les etiquetes codificades
```

```
  integer_encoded = np.array(integer_encoded).reshape(-1, 1)
  one_hot_encoded = one_hot_encoder.fit_transform(integer_encoded)
  input_features.append(one_hot_encoded.toarray())

np.set_printoptions(threshold=40)
input_features = np.stack(input_features)
print("Example sequence\n-----------------------")
print('DNA Sequence #1:\n',sequences[0][:10],'...',sequences[0][-10:])
print('One hot encoding of Sequence #1:\n',input_features[0].T)
```

```
Example sequence
-----------------------
DNA Sequence #1:
 CCGAGGGCTA ... CGCGGACACC
One hot encoding of Sequence #1:
 [[0. 0. 0. ... 1. 0. 0.]
 [1. 1. 0. ... 0. 1. 1.]
 [0. 0. 1. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```

```
# Codi 3

LABELS_URL = 'https://raw.githubusercontent.com/abidlabs/deep-learning
-genomics-primer/master/labels.txt'

labels = requests.get(LABELS_URL).text.split('\n')
labels = list(filter(None, labels))
# es neteja la base de dades dels valors buits}

one_hot_encoder = OneHotEncoder(categories='auto')
labels = np.array(labels).reshape(-1, 1)
input_labels = one_hot_encoder.fit_transform(labels).toarray()

print('Labels:\n',labels.T)
print('One-hot encoded labels:\n',input_labels.T)
```

```
Labels:
 [['0' '0' '0' ... '0' '1' '1']]
One-hot encoded labels:
 [[1. 1. 1. ... 1. 0. 0.]
 [0. 0. 0. ... 0. 1. 1.]]
```

```
# Codi 4

from sklearn.model_selection import train_test_split

train_features, test_features, train_labels, test_labels= train_test_s
plit(
    input_features, input_labels, test_size=0.25, random_state=42)
```

Construcció de l'arquitectura i entrenament

```
# Codi 5

# Es carreguen les funcions que es necessitaran
# per fer el model de deep learning
from tensorflow.keras.layers import Conv1D, Dense, MaxPooling1D, Flatt
en
# Es carrega el tipus de model que es durà a terme.
from tensorflow.keras.models import Sequential

# A contiuació es construeix el model a partir d'afegir
# (model.add) capes al model sequential.
model = Sequential()
model.add(Conv1D(filters=32, kernel_size=12,
                 input_shape=(train_features.shape[1], 4)))
model.add(MaxPooling1D(pool_size=4))
model.add(Flatten())
model.add(Dense(16, activation='relu'))
model.add(Dense(2, activation='softmax'))
```

```
# Codi 6

# Es compila el model a partir de la funció model.compile
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['
binary_accuracy'])

# Mostra els elements del model
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_7 (Conv1D)            (None, 39, 32)            1568
_________________________________________________________________
max_pooling1d_7 (MaxPooling1 (None, 9, 32)             0
_________________________________________________________________
flatten_7 (Flatten)          (None, 288)               0
_________________________________________________________________
dense_14 (Dense)             (None, 16)                4624
_________________________________________________________________
dense_15 (Dense)             (None, 2)                 34
=================================================================
Total params: 6,226
Trainable params: 6,226
Non-trainable params: 0
_________________________________________________________________
```

```
# Codi 7

history = model.fit(train_features, train_labels,
                    epochs=50, verbose=1, validation_split=0.25)
```

```
plt.figure()
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Funció de pèrdua')
plt.ylabel('pèrdua')
plt.xlabel('epoch')
plt.legend(['Dades entrenament', 'Dades validació'])
plt.show()
```

```
Train on 1125 samples, validate on 375 samples
Epoch 1/50
1125/1125 [==============================] - 1s 535us/sample - loss: 0
.6972 - binary_accuracy: 0.5191 - val_loss: 0.6482 - val_binary_accura
cy: 0.6133
Epoch 2/50
1125/1125 [==============================] - 0s 117us/sample - loss: 0
.5874 - binary_accuracy: 0.7618 - val_loss: 0.5123 - val_binary_accura
cy: 0.7413
Epoch 3/50
1125/1125 [==============================] - 0s 115us/sample - loss: 0
.4299 - binary_accuracy: 0.8409 - val_loss: 0.3753 - val_binary_accura
cy: 0.8373
Epoch 4/50
1125/1125 [==============================] - 0s 118us/sample - loss: 0
.3157 - binary_accuracy: 0.8907 - val_loss: 0.2786 - val_binary_accura
cy: 0.8960
Epoch 5/50
1125/1125 [==============================] - 0s 117us/sample - loss: 0
.2411 - binary_accuracy: 0.9244 - val_loss: 0.2360 - val_binary_accura
cy: 0.9253
Epoch 6/50
1125/1125 [==============================] - 0s 116us/sample - loss: 0
.1867 - binary_accuracy: 0.9458 - val_loss: 0.1982 - val_binary_accura
cy: 0.9280
Epoch 7/50
1125/1125 [==============================] - 0s 115us/sample - loss: 0
.1419 - binary_accuracy: 0.9662 - val_loss: 0.1718 - val_binary_accura
cy: 0.9387
Epoch 8/50
1125/1125 [==============================] - 0s 116us/sample - loss: 0
.1125 - binary_accuracy: 0.9760 - val_loss: 0.1539 - val_binary_accura
cy: 0.9413
Epoch 9/50
1125/1125 [==============================] - 0s 117us/sample - loss: 0
.0998 - binary_accuracy: 0.9742 - val_loss: 0.1820 - val_binary_accura
cy: 0.9387
Epoch 10/50
1125/1125 [==============================] - 0s 117us/sample - loss: 0
.0774 - binary_accuracy: 0.9804 - val_loss: 0.1524 - val_binary_accura
cy: 0.9440
Epoch 11/50
1125/1125 [==============================] - 0s 115us/sample - loss: 0
.0649 - binary_accuracy: 0.9849 - val_loss: 0.1474 - val_binary_accura
```

```
cy: 0.9440
Epoch 12/50
1125/1125 [==============================] - 0s 118us/sample - loss: 0
.0530 - binary_accuracy: 0.9876 - val_loss: 0.1221 - val_binary_accura
cy: 0.9520
Epoch 13/50
1125/1125 [==============================] - 0s 115us/sample - loss: 0
.0436 - binary_accuracy: 0.9956 - val_loss: 0.1203 - val_binary_accura
cy: 0.9573
Epoch 14/50
1125/1125 [==============================] - 0s 118us/sample - loss: 0
.0371 - binary_accuracy: 0.9920 - val_loss: 0.1900 - val_binary_accura
cy: 0.9440
Epoch 15/50
1125/1125 [==============================] - 0s 117us/sample - loss: 0
.0295 - binary_accuracy: 0.9973 - val_loss: 0.1361 - val_binary_accura
cy: 0.9547
Epoch 16/50
1125/1125 [==============================] - 0s 117us/sample - loss: 0
.0313 - binary_accuracy: 0.9956 - val_loss: 0.1210 - val_binary_accura
cy: 0.9547
Epoch 17/50
1125/1125 [==============================] - 0s 116us/sample - loss: 0
.0173 - binary_accuracy: 1.0000 - val_loss: 0.1273 - val_binary_accura
cy: 0.9600
Epoch 18/50
1125/1125 [==============================] - 0s 158us/sample - loss: 0
.0167 - binary_accuracy: 1.0000 - val_loss: 0.1138 - val_binary_accura
cy: 0.9600
Epoch 19/50
1125/1125 [==============================] - 0s 239us/sample - loss: 0
.0128 - binary_accuracy: 1.0000 - val_loss: 0.1275 - val_binary_accura
cy: 0.9600
Epoch 20/50
1125/1125 [==============================] - 0s 175us/sample - loss: 0
.0119 - binary_accuracy: 1.0000 - val_loss: 0.1160 - val_binary_accura
cy: 0.9600
Epoch 21/50
1125/1125 [==============================] - 0s 150us/sample - loss: 0
.0096 - binary_accuracy: 1.0000 - val_loss: 0.1212 - val_binary_accura
cy: 0.9600
Epoch 22/50
1125/1125 [==============================] - 0s 181us/sample - loss: 0
.0089 - binary_accuracy: 1.0000 - val_loss: 0.1218 - val_binary_accura
cy: 0.9600
Epoch 23/50
1125/1125 [==============================] - 0s 160us/sample - loss: 0
.0074 - binary_accuracy: 1.0000 - val_loss: 0.1178 - val_binary_accura
cy: 0.9600
Epoch 24/50
1125/1125 [==============================] - 0s 151us/sample - loss: 0
.0067 - binary_accuracy: 1.0000 - val_loss: 0.1231 - val_binary_accura
cy: 0.9600
Epoch 25/50
```

```
1125/1125 [==============================] - 0s 157us/sample - loss: 0
.0059 - binary_accuracy: 1.0000 - val_loss: 0.1172 - val_binary_accura
cy: 0.9600
Epoch 26/50
1125/1125 [==============================] - 0s 156us/sample - loss: 0
.0058 - binary_accuracy: 1.0000 - val_loss: 0.1112 - val_binary_accura
cy: 0.9653
Epoch 27/50
1125/1125 [==============================] - 0s 122us/sample - loss: 0
.0051 - binary_accuracy: 1.0000 - val_loss: 0.1212 - val_binary_accura
cy: 0.9600
Epoch 28/50
1125/1125 [==============================] - 0s 120us/sample - loss: 0
.0043 - binary_accuracy: 1.0000 - val_loss: 0.1131 - val_binary_accura
cy: 0.9653
Epoch 29/50
1125/1125 [==============================] - 0s 118us/sample - loss: 0
.0040 - binary_accuracy: 1.0000 - val_loss: 0.1154 - val_binary_accura
cy: 0.9653
Epoch 30/50
1125/1125 [==============================] - 0s 123us/sample - loss: 0
.0036 - binary_accuracy: 1.0000 - val_loss: 0.1210 - val_binary_accura
cy: 0.9600
Epoch 31/50
1125/1125 [==============================] - 0s 133us/sample - loss: 0
.0032 - binary_accuracy: 1.0000 - val_loss: 0.1184 - val_binary_accura
cy: 0.9653
Epoch 32/50
1125/1125 [==============================] - 0s 124us/sample - loss: 0
.0031 - binary_accuracy: 1.0000 - val_loss: 0.1168 - val_binary_accura
cy: 0.9653
Epoch 33/50
1125/1125 [==============================] - 0s 192us/sample - loss: 0
.0027 - binary_accuracy: 1.0000 - val_loss: 0.1128 - val_binary_accura
cy: 0.9653
Epoch 34/50
1125/1125 [==============================] - 0s 156us/sample - loss: 0
.0025 - binary_accuracy: 1.0000 - val_loss: 0.1181 - val_binary_accura
cy: 0.9653
Epoch 35/50
1125/1125 [==============================] - 0s 131us/sample - loss: 0
.0023 - binary_accuracy: 1.0000 - val_loss: 0.1254 - val_binary_accura
cy: 0.9627
Epoch 36/50
1125/1125 [==============================] - 0s 120us/sample - loss: 0
.0023 - binary_accuracy: 1.0000 - val_loss: 0.1187 - val_binary_accura
cy: 0.9653
Epoch 37/50
1125/1125 [==============================] - 0s 105us/sample - loss: 0
.0020 - binary_accuracy: 1.0000 - val_loss: 0.1224 - val_binary_accura
cy: 0.9653
Epoch 38/50
1125/1125 [==============================] - 0s 103us/sample - loss: 0
.0019 - binary_accuracy: 1.0000 - val_loss: 0.1190 - val_binary_accura
```

```
cy: 0.9653
Epoch 39/50
1125/1125 [==============================] - 0s 111us/sample - loss: 0
.0018 - binary_accuracy: 1.0000 - val_loss: 0.1168 - val_binary_accura
cy: 0.9653
Epoch 40/50
1125/1125 [==============================] - 0s 115us/sample - loss: 0
.0017 - binary_accuracy: 1.0000 - val_loss: 0.1254 - val_binary_accura
cy: 0.9653
Epoch 41/50
1125/1125 [==============================] - 0s 188us/sample - loss: 0
.0016 - binary_accuracy: 1.0000 - val_loss: 0.1246 - val_binary_accura
cy: 0.9653
Epoch 42/50
1125/1125 [==============================] - 0s 142us/sample - loss: 0
.0014 - binary_accuracy: 1.0000 - val_loss: 0.1223 - val_binary_accura
cy: 0.9653
Epoch 43/50
1125/1125 [==============================] - 0s 107us/sample - loss: 0
.0014 - binary_accuracy: 1.0000 - val_loss: 0.1295 - val_binary_accura
cy: 0.9653
Epoch 44/50
1125/1125 [==============================] - 0s 104us/sample - loss: 0
.0013 - binary_accuracy: 1.0000 - val_loss: 0.1238 - val_binary_accura
cy: 0.9653
Epoch 45/50
1125/1125 [==============================] - 0s 191us/sample - loss: 0
.0012 - binary_accuracy: 1.0000 - val_loss: 0.1224 - val_binary_accura
cy: 0.9653
Epoch 46/50
1125/1125 [==============================] - 0s 128us/sample - loss: 0
.0012 - binary_accuracy: 1.0000 - val_loss: 0.1315 - val_binary_accura
cy: 0.9653
Epoch 47/50
1125/1125 [==============================] - 0s 117us/sample - loss: 0
.0011 - binary_accuracy: 1.0000 - val_loss: 0.1211 - val_binary_accura
cy: 0.9653
Epoch 48/50
1125/1125 [==============================] - 0s 177us/sample - loss: 0
.0010 - binary_accuracy: 1.0000 - val_loss: 0.1266 - val_binary_accura
cy: 0.9653
Epoch 49/50
1125/1125 [==============================] - 0s 184us/sample - loss: 9
.5087e-04 - binary_accuracy: 1.0000 - val_loss: 0.1223 - val_binary_ac
curacy: 0.9653
Epoch 50/50
1125/1125 [==============================] - 0s 129us/sample - loss: 9
.0443e-04 - binary_accuracy: 1.0000 - val_loss: 0.1260 - val_binary_ac
curacy: 0.9653
```

```
# Codi 8

plt.figure()
plt.plot(history.history['binary_accuracy'])
plt.plot(history.history['val_binary_accuracy'])
plt.title('Accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['Dades entrenament', 'dades validació'])
plt.show()
```

Avaluació

```
# Codi 9

from sklearn.metrics import confusion_matrix
import itertools

predicted_labels = model.predict(np.stack(test_features))
cm = confusion_matrix(np.argmax(test_labels, axis=1),
                      np.argmax(predicted_labels, axis=1))
print('Matriu de confusió:\n',cm)

cm = cm.astype('float') / cm.sum(axis = 1)[:, np.newaxis]

plt.imshow(cm, cmap=plt.cm.Blues)
plt.title('Matriu de confusió normalitzada')
plt.colorbar()
plt.xlabel('Nivells reals')
plt.ylabel('Nivells predits')
plt.xticks([0, 1]); plt.yticks([0, 1])
plt.grid('off')
for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
    plt.text(j, i, format(cm[i, j], '.2f'),
             horizontalalignment='center',
             color='white' if cm[i, j] > 0.5 else 'black')

Matriu de confusió:
 [[250   9]
 [  6 235]]
```

Interpretació

```
import tensorflow.keras.backend as K

def compute_salient_bases(model, x):
  input_tensors = [model.input]
  gradients = model.optimizer.get_gradients(model.output[0][1], model.
input)
  compute_gradients = K.function(inputs = input_tensors, outputs = gra
dients)
```

```
  x_value = np.expand_dims(x, axis=0)
  gradients = compute_gradients([x_value])[0][0]
  sal = np.clip(np.sum(np.multiply(gradients,x), axis=1),a_min=0, a_ma
x=None)
  return sal

sequence_index = 1999  #  Es pot canviar el valor de la seqüència.
sal = compute_salient_bases(model, input_features[sequence_index])

plt.figure(figsize=[16,5])
barlist = plt.bar(np.arange(len(sal)), sal)
[barlist[i].set_color('C1') for i in range(5,17)]
plt.xlabel('Bases')
plt.ylabel('Magnitud del saliency values')
plt.xticks(np.arange(len(sal)), list(sequences[sequence_index]));
plt.title('Saliency map per bases en una seqüència positica'
          ' (green indicates the actual bases in motif)');
```

Nous problemes

Canvi d'alguns paràmetres

```
# Codi 10

model = Sequential()
model.add(Conv1D(filters=32, kernel_size=12,
                 input_shape=(train_features.shape[1], 4)))
model.add(MaxPooling1D(pool_size=4))
model.add(Flatten())
model.add(Dense(16, activation='relu'))
model.add(Dense(2, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer='sgd',
              metrics=['binary_accuracy'])


mod1 = model.fit(train_features, train_labels,
                    epochs=30, verbose=1, validation_split=0.25)
```

```
Matriu de confusió:
 [[182  77]
 [ 25 216]]
```

```
# Codi 10.2

predicted_labels = model.predict(np.stack(test_features))
cm = confusion_matrix(np.argmax(test_labels, axis=1),
                      np.argmax(predicted_labels, axis=1))
print('Matriu de confusió:\n',cm)

history.mod1['binary_accuracy'][29]
```

```
Matriu de confusió:
 [[182  77]
 [ 25 216]]
```

```
# Codi 11

model = Sequential()
model.add(Conv1D(filters=32, kernel_size=12,
                 input_shape=(train_features.shape[1], 4)))
model.add(MaxPooling1D(pool_size=4))
model.add(Flatten())
model.add(Dense(16, activation='relu'))
model.add(Dense(2, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer='sgd',
              metrics=['binary_accuracy'])

mod2 = model.fit(train_features, train_labels,
                   epochs=50, verbose=0, validation_split=0.25)
```

```
Train on 1125 samples, validate on 375 samples
Epoch 1/50
1125/1125 [==============================] - 1s 503us/sample - loss: 0
.7003 - binary_accuracy: 0.4902 - val_loss: 0.6953 - val_binary_accura
cy: 0.5133
Epoch 2/50
1125/1125 [==============================] - 0s 122us/sample - loss: 0
.6939 - binary_accuracy: 0.5093 - val_loss: 0.6879 - val_binary_accura
cy: 0.5493
Epoch 3/50
1125/1125 [==============================] - 0s 133us/sample - loss: 0
.6886 - binary_accuracy: 0.5338 - val_loss: 0.6829 - val_binary_accura
cy: 0.5653
Epoch 4/50
1125/1125 [==============================] - 0s 137us/sample - loss: 0
.6836 - binary_accuracy: 0.5609 - val_loss: 0.6767 - val_binary_accura
cy: 0.5800
Epoch 5/50
1125/1125 [==============================] - 0s 132us/sample - loss: 0
.6783 - binary_accuracy: 0.5849 - val_loss: 0.6703 - val_binary_accura
cy: 0.6120
Epoch 6/50
1125/1125 [==============================] - 0s 143us/sample - loss: 0
.6723 - binary_accuracy: 0.6022 - val_loss: 0.6636 - val_binary_accura
cy: 0.6387
Epoch 7/50
1125/1125 [==============================] - 0s 126us/sample - loss: 0
.6645 - binary_accuracy: 0.6342 - val_loss: 0.6550 - val_binary_accura
cy: 0.6640
Epoch 8/50
1125/1125 [==============================] - 0s 131us/sample - loss: 0
.6553 - binary_accuracy: 0.6547 - val_loss: 0.6478 - val_binary_accura
```

```
cy: 0.6653
Epoch 9/50
1125/1125 [==============================] - 0s 126us/sample - loss: 0
.6461 - binary_accuracy: 0.6564 - val_loss: 0.6357 - val_binary_accura
cy: 0.7053
Epoch 10/50
1125/1125 [==============================] - 0s 123us/sample - loss: 0
.6363 - binary_accuracy: 0.6951 - val_loss: 0.6248 - val_binary_accura
cy: 0.7147
Epoch 11/50
1125/1125 [==============================] - 0s 106us/sample - loss: 0
.6253 - binary_accuracy: 0.7111 - val_loss: 0.6126 - val_binary_accura
cy: 0.7240
Epoch 12/50
1125/1125 [==============================] - 0s 113us/sample - loss: 0
.6130 - binary_accuracy: 0.7258 - val_loss: 0.6000 - val_binary_accura
cy: 0.7333
Epoch 13/50
1125/1125 [==============================] - 0s 118us/sample - loss: 0
.6007 - binary_accuracy: 0.7413 - val_loss: 0.5872 - val_binary_accura
cy: 0.7400
Epoch 14/50
1125/1125 [==============================] - 0s 120us/sample - loss: 0
.5868 - binary_accuracy: 0.7529 - val_loss: 0.5715 - val_binary_accura
cy: 0.7613
Epoch 15/50
1125/1125 [==============================] - 0s 123us/sample - loss: 0
.5725 - binary_accuracy: 0.7676 - val_loss: 0.5563 - val_binary_accura
cy: 0.7680
Epoch 16/50
1125/1125 [==============================] - 0s 114us/sample - loss: 0
.5578 - binary_accuracy: 0.7773 - val_loss: 0.5415 - val_binary_accura
cy: 0.7733
Epoch 17/50
1125/1125 [==============================] - 0s 199us/sample - loss: 0
.5423 - binary_accuracy: 0.7804 - val_loss: 0.5236 - val_binary_accura
cy: 0.7853
Epoch 18/50
1125/1125 [==============================] - 0s 176us/sample - loss: 0
.5254 - binary_accuracy: 0.7902 - val_loss: 0.5068 - val_binary_accura
cy: 0.8040
Epoch 19/50
1125/1125 [==============================] - 0s 161us/sample - loss: 0
.5080 - binary_accuracy: 0.7929 - val_loss: 0.4898 - val_binary_accura
cy: 0.8133
Epoch 20/50
1125/1125 [==============================] - 0s 182us/sample - loss: 0
.4913 - binary_accuracy: 0.8107 - val_loss: 0.4743 - val_binary_accura
cy: 0.8173
Epoch 21/50
1125/1125 [==============================] - 0s 186us/sample - loss: 0
.4755 - binary_accuracy: 0.8111 - val_loss: 0.4621 - val_binary_accura
cy: 0.7973
Epoch 22/50
```

```
1125/1125 [==============================] - 0s 144us/sample - loss: 0
.4594 - binary_accuracy: 0.8191 - val_loss: 0.4406 - val_binary_accura
cy: 0.8307
Epoch 23/50
1125/1125 [==============================] - 0s 159us/sample - loss: 0
.4420 - binary_accuracy: 0.8298 - val_loss: 0.4376 - val_binary_accura
cy: 0.8213
Epoch 24/50
1125/1125 [==============================] - 0s 119us/sample - loss: 0
.4271 - binary_accuracy: 0.8333 - val_loss: 0.4104 - val_binary_accura
cy: 0.8467
Epoch 25/50
1125/1125 [==============================] - 0s 119us/sample - loss: 0
.4108 - binary_accuracy: 0.8444 - val_loss: 0.3969 - val_binary_accura
cy: 0.8493
Epoch 26/50
1125/1125 [==============================] - 0s 118us/sample - loss: 0
.3955 - binary_accuracy: 0.8524 - val_loss: 0.3820 - val_binary_accura
cy: 0.8507
Epoch 27/50
1125/1125 [==============================] - 0s 125us/sample - loss: 0
.3810 - binary_accuracy: 0.8569 - val_loss: 0.3706 - val_binary_accura
cy: 0.8480
Epoch 28/50
1125/1125 [==============================] - 0s 126us/sample - loss: 0
.3670 - binary_accuracy: 0.8684 - val_loss: 0.3675 - val_binary_accura
cy: 0.8560
Epoch 29/50
1125/1125 [==============================] - 0s 119us/sample - loss: 0
.3551 - binary_accuracy: 0.8680 - val_loss: 0.3461 - val_binary_accura
cy: 0.8587
Epoch 30/50
1125/1125 [==============================] - 0s 124us/sample - loss: 0
.3412 - binary_accuracy: 0.8756 - val_loss: 0.3538 - val_binary_accura
cy: 0.8400
Epoch 31/50
1125/1125 [==============================] - 0s 196us/sample - loss: 0
.3293 - binary_accuracy: 0.8844 - val_loss: 0.3233 - val_binary_accura
cy: 0.8707
Epoch 32/50
1125/1125 [==============================] - 0s 188us/sample - loss: 0
.3149 - binary_accuracy: 0.8907 - val_loss: 0.3138 - val_binary_accura
cy: 0.8707
Epoch 33/50
1125/1125 [==============================] - 0s 158us/sample - loss: 0
.3045 - binary_accuracy: 0.8893 - val_loss: 0.3045 - val_binary_accura
cy: 0.8853
Epoch 34/50
1125/1125 [==============================] - 0s 163us/sample - loss: 0
.2922 - binary_accuracy: 0.8978 - val_loss: 0.2947 - val_binary_accura
cy: 0.8880
Epoch 35/50
1125/1125 [==============================] - 0s 118us/sample - loss: 0
.2810 - binary_accuracy: 0.9018 - val_loss: 0.2887 - val_binary_accura
```

```
cy: 0.8800
Epoch 36/50
1125/1125 [==============================] - 0s 127us/sample - loss: 0
.2726 - binary_accuracy: 0.9098 - val_loss: 0.2792 - val_binary_accura
cy: 0.8880
Epoch 37/50
1125/1125 [==============================] - 0s 113us/sample - loss: 0
.2616 - binary_accuracy: 0.9178 - val_loss: 0.2740 - val_binary_accura
cy: 0.9067
Epoch 38/50
1125/1125 [==============================] - 0s 109us/sample - loss: 0
.2515 - binary_accuracy: 0.9173 - val_loss: 0.2642 - val_binary_accura
cy: 0.9080
Epoch 39/50
1125/1125 [==============================] - 0s 115us/sample - loss: 0
.2419 - binary_accuracy: 0.9191 - val_loss: 0.2556 - val_binary_accura
cy: 0.9147
Epoch 40/50
1125/1125 [==============================] - 0s 125us/sample - loss: 0
.2336 - binary_accuracy: 0.9271 - val_loss: 0.2480 - val_binary_accura
cy: 0.9120
Epoch 41/50
1125/1125 [==============================] - 0s 123us/sample - loss: 0
.2268 - binary_accuracy: 0.9267 - val_loss: 0.2525 - val_binary_accura
cy: 0.9133
Epoch 42/50
1125/1125 [==============================] - 0s 111us/sample - loss: 0
.2179 - binary_accuracy: 0.9347 - val_loss: 0.2500 - val_binary_accura
cy: 0.9160
Epoch 43/50
1125/1125 [==============================] - 0s 120us/sample - loss: 0
.2117 - binary_accuracy: 0.9369 - val_loss: 0.2443 - val_binary_accura
cy: 0.8960
Epoch 44/50
1125/1125 [==============================] - 0s 113us/sample - loss: 0
.2063 - binary_accuracy: 0.9396 - val_loss: 0.2244 - val_binary_accura
cy: 0.9227
Epoch 45/50
1125/1125 [==============================] - 0s 128us/sample - loss: 0
.1962 - binary_accuracy: 0.9391 - val_loss: 0.2554 - val_binary_accura
cy: 0.9027
Epoch 46/50
1125/1125 [==============================] - 0s 132us/sample - loss: 0
.1912 - binary_accuracy: 0.9453 - val_loss: 0.2201 - val_binary_accura
cy: 0.9093
Epoch 47/50
1125/1125 [==============================] - 0s 136us/sample - loss: 0
.1860 - binary_accuracy: 0.9493 - val_loss: 0.2269 - val_binary_accura
cy: 0.9253
Epoch 48/50
1125/1125 [==============================] - 0s 145us/sample - loss: 0
.1798 - binary_accuracy: 0.9458 - val_loss: 0.2053 - val_binary_accura
cy: 0.9347
Epoch 49/50
```

```
1125/1125 [==============================] - 0s 137us/sample - loss: 0
.1742 - binary_accuracy: 0.9516 - val_loss: 0.2022 - val_binary_accura
cy: 0.9360
Epoch 50/50
1125/1125 [==============================] - 0s 200us/sample - loss: 0
.1707 - binary_accuracy: 0.9524 - val_loss: 0.1965 - val_binary_accura
cy: 0.9413
```

```
# Codi 11.2
predicted_labels = model.predict(np.stack(test_features))
cm = confusion_matrix(np.argmax(test_labels, axis=1),
                      np.argmax(predicted_labels, axis=1))
print('Matriu de confusió:\n',cm)

history.mod2['binary_accuracy'][49]
```

```
Matriu de confusió:
 [[242  17]
 [ 15 226]]
```

```
# Codi 12

model = Sequential()
model.add(Conv1D(filters=32, kernel_size=12,
                 input_shape=(train_features.shape[1], 4)))
model.add(MaxPooling1D(pool_size=4))
model.add(Flatten())
model.add(Dense(16, activation='relu'))
model.add(Dense(2, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['binary_accuracy'])

mod = model.fit(train_features, train_labels,
                    epochs=50, verbose=1, validation_split=0.25)
```

```
Train on 1125 samples, validate on 375 samples
Epoch 1/50
1125/1125 [==============================] - 1s 554us/sample - loss: 0
.6786 - binary_accuracy: 0.5622 - val_loss: 0.6370 - val_binary_accura
cy: 0.7347
Epoch 2/50
1125/1125 [==============================] - 0s 119us/sample - loss: 0
.5744 - binary_accuracy: 0.7760 - val_loss: 0.4927 - val_binary_accura
cy: 0.8307
Epoch 3/50
1125/1125 [==============================] - 0s 130us/sample - loss: 0
.4118 - binary_accuracy: 0.8511 - val_loss: 0.3292 - val_binary_accura
cy: 0.8933
Epoch 4/50
1125/1125 [==============================] - 0s 127us/sample - loss: 0
.2825 - binary_accuracy: 0.9089 - val_loss: 0.2542 - val_binary_accura
cy: 0.9240
```

```
Epoch 5/50
1125/1125 [==============================] - 0s 122us/sample - loss: 0
.2122 - binary_accuracy: 0.9293 - val_loss: 0.2025 - val_binary_accura
cy: 0.9387
Epoch 6/50
1125/1125 [==============================] - 0s 122us/sample - loss: 0
.1620 - binary_accuracy: 0.9618 - val_loss: 0.1710 - val_binary_accura
cy: 0.9533
Epoch 7/50
1125/1125 [==============================] - 0s 122us/sample - loss: 0
.1269 - binary_accuracy: 0.9716 - val_loss: 0.1480 - val_binary_accura
cy: 0.9547
Epoch 8/50
1125/1125 [==============================] - 0s 121us/sample - loss: 0
.1112 - binary_accuracy: 0.9747 - val_loss: 0.1321 - val_binary_accura
cy: 0.9627
Epoch 9/50
1125/1125 [==============================] - 0s 122us/sample - loss: 0
.0887 - binary_accuracy: 0.9782 - val_loss: 0.1393 - val_binary_accura
cy: 0.9627
Epoch 10/50
1125/1125 [==============================] - 0s 121us/sample - loss: 0
.0734 - binary_accuracy: 0.9849 - val_loss: 0.1302 - val_binary_accura
cy: 0.9653
Epoch 11/50
1125/1125 [==============================] - 0s 119us/sample - loss: 0
.0603 - binary_accuracy: 0.9916 - val_loss: 0.1407 - val_binary_accura
cy: 0.9560
Epoch 12/50
1125/1125 [==============================] - 0s 152us/sample - loss: 0
.0657 - binary_accuracy: 0.9836 - val_loss: 0.1170 - val_binary_accura
cy: 0.9707
Epoch 13/50
1125/1125 [==============================] - 0s 155us/sample - loss: 0
.0432 - binary_accuracy: 0.9942 - val_loss: 0.1074 - val_binary_accura
cy: 0.9747
Epoch 14/50
1125/1125 [==============================] - 0s 146us/sample - loss: 0
.0374 - binary_accuracy: 0.9964 - val_loss: 0.1086 - val_binary_accura
cy: 0.9733
Epoch 15/50
1125/1125 [==============================] - 0s 153us/sample - loss: 0
.0307 - binary_accuracy: 0.9960 - val_loss: 0.1057 - val_binary_accura
cy: 0.9760
Epoch 16/50
1125/1125 [==============================] - 0s 126us/sample - loss: 0
.0253 - binary_accuracy: 0.9982 - val_loss: 0.0967 - val_binary_accura
cy: 0.9760
Epoch 17/50
1125/1125 [==============================] - 0s 126us/sample - loss: 0
.0211 - binary_accuracy: 0.9991 - val_loss: 0.1188 - val_binary_accura
cy: 0.9680
Epoch 18/50
1125/1125 [==============================] - 0s 207us/sample - loss: 0
```

```
.0200 - binary_accuracy: 1.0000 - val_loss: 0.0913 - val_binary_accura
cy: 0.9800
Epoch 19/50
1125/1125 [==============================] - 0s 171us/sample - loss: 0
.0199 - binary_accuracy: 0.9996 - val_loss: 0.1006 - val_binary_accura
cy: 0.9787
Epoch 20/50
1125/1125 [==============================] - 0s 203us/sample - loss: 0
.0141 - binary_accuracy: 1.0000 - val_loss: 0.1049 - val_binary_accura
cy: 0.9747
Epoch 21/50
1125/1125 [==============================] - 0s 205us/sample - loss: 0
.0119 - binary_accuracy: 1.0000 - val_loss: 0.1038 - val_binary_accura
cy: 0.9787
Epoch 22/50
1125/1125 [==============================] - 0s 178us/sample - loss: 0
.0107 - binary_accuracy: 1.0000 - val_loss: 0.0942 - val_binary_accura
cy: 0.9787
Epoch 23/50
1125/1125 [==============================] - 0s 127us/sample - loss: 0
.0091 - binary_accuracy: 1.0000 - val_loss: 0.0904 - val_binary_accura
cy: 0.9773
Epoch 24/50
1125/1125 [==============================] - 0s 125us/sample - loss: 0
.0082 - binary_accuracy: 1.0000 - val_loss: 0.0954 - val_binary_accura
cy: 0.9787
Epoch 25/50
1125/1125 [==============================] - 0s 129us/sample - loss: 0
.0075 - binary_accuracy: 1.0000 - val_loss: 0.0963 - val_binary_accura
cy: 0.9787
Epoch 26/50
1125/1125 [==============================] - 0s 136us/sample - loss: 0
.0065 - binary_accuracy: 1.0000 - val_loss: 0.1034 - val_binary_accura
cy: 0.9787
Epoch 27/50
1125/1125 [==============================] - 0s 132us/sample - loss: 0
.0059 - binary_accuracy: 1.0000 - val_loss: 0.1025 - val_binary_accura
cy: 0.9787
Epoch 28/50
1125/1125 [==============================] - 0s 129us/sample - loss: 0
.0053 - binary_accuracy: 1.0000 - val_loss: 0.1055 - val_binary_accura
cy: 0.9787
Epoch 29/50
1125/1125 [==============================] - 0s 126us/sample - loss: 0
.0048 - binary_accuracy: 1.0000 - val_loss: 0.1109 - val_binary_accura
cy: 0.9773
Epoch 30/50
1125/1125 [==============================] - 0s 128us/sample - loss: 0
.0046 - binary_accuracy: 1.0000 - val_loss: 0.1014 - val_binary_accura
cy: 0.9787
Epoch 31/50
1125/1125 [==============================] - 0s 138us/sample - loss: 0
.0041 - binary_accuracy: 1.0000 - val_loss: 0.1195 - val_binary_accura
cy: 0.9760
```

```
Epoch 32/50
1125/1125 [==============================] - 0s 250us/sample - loss: 0
.0038 - binary_accuracy: 1.0000 - val_loss: 0.1082 - val_binary_accura
cy: 0.9787
Epoch 33/50
1125/1125 [==============================] - 0s 181us/sample - loss: 0
.0037 - binary_accuracy: 1.0000 - val_loss: 0.1022 - val_binary_accura
cy: 0.9787
Epoch 34/50
1125/1125 [==============================] - 0s 180us/sample - loss: 0
.0033 - binary_accuracy: 1.0000 - val_loss: 0.1079 - val_binary_accura
cy: 0.9787
Epoch 35/50
1125/1125 [==============================] - 0s 124us/sample - loss: 0
.0030 - binary_accuracy: 1.0000 - val_loss: 0.1063 - val_binary_accura
cy: 0.9787
Epoch 36/50
1125/1125 [==============================] - 0s 132us/sample - loss: 0
.0029 - binary_accuracy: 1.0000 - val_loss: 0.1109 - val_binary_accura
cy: 0.9787
Epoch 37/50
1125/1125 [==============================] - 0s 115us/sample - loss: 0
.0026 - binary_accuracy: 1.0000 - val_loss: 0.1086 - val_binary_accura
cy: 0.9787
Epoch 38/50
1125/1125 [==============================] - 0s 116us/sample - loss: 0
.0024 - binary_accuracy: 1.0000 - val_loss: 0.1047 - val_binary_accura
cy: 0.9787
Epoch 39/50
1125/1125 [==============================] - 0s 117us/sample - loss: 0
.0023 - binary_accuracy: 1.0000 - val_loss: 0.1198 - val_binary_accura
cy: 0.9773
Epoch 40/50
1125/1125 [==============================] - 0s 129us/sample - loss: 0
.0022 - binary_accuracy: 1.0000 - val_loss: 0.1097 - val_binary_accura
cy: 0.9787
Epoch 41/50
1125/1125 [==============================] - 0s 125us/sample - loss: 0
.0019 - binary_accuracy: 1.0000 - val_loss: 0.1034 - val_binary_accura
cy: 0.9787
Epoch 42/50
1125/1125 [==============================] - 0s 126us/sample - loss: 0
.0019 - binary_accuracy: 1.0000 - val_loss: 0.1131 - val_binary_accura
cy: 0.9787
Epoch 43/50
1125/1125 [==============================] - 0s 123us/sample - loss: 0
.0017 - binary_accuracy: 1.0000 - val_loss: 0.1041 - val_binary_accura
cy: 0.9787
Epoch 44/50
1125/1125 [==============================] - 0s 116us/sample - loss: 0
.0016 - binary_accuracy: 1.0000 - val_loss: 0.1161 - val_binary_accura
cy: 0.9787
Epoch 45/50
1125/1125 [==============================] - 0s 240us/sample - loss: 0
```

```
.0015 - binary_accuracy: 1.0000 - val_loss: 0.1134 - val_binary_accura
cy: 0.9787
Epoch 46/50
1125/1125 [==============================] - 0s 123us/sample - loss: 0
.0014 - binary_accuracy: 1.0000 - val_loss: 0.1129 - val_binary_accura
cy: 0.9787
Epoch 47/50
1125/1125 [==============================] - 0s 192us/sample - loss: 0
.0014 - binary_accuracy: 1.0000 - val_loss: 0.1084 - val_binary_accura
cy: 0.9787
Epoch 48/50
1125/1125 [==============================] - 0s 156us/sample - loss: 0
.0013 - binary_accuracy: 1.0000 - val_loss: 0.1183 - val_binary_accura
cy: 0.9787
Epoch 49/50
1125/1125 [==============================] - 0s 135us/sample - loss: 0
.0013 - binary_accuracy: 1.0000 - val_loss: 0.1172 - val_binary_accura
cy: 0.9787
Epoch 50/50
1125/1125 [==============================] - 0s 120us/sample - loss: 0
.0012 - binary_accuracy: 1.0000 - val_loss: 0.1188 - val_binary_accura
cy: 0.9787
```

```
# Codi 12.2
predicted_labels = model.predict(np.stack(test_features))
cm = confusion_matrix(np.argmax(test_labels, axis=1),
                      np.argmax(predicted_labels, axis=1))
print('Matriu de confusió:\n',cm)

mod2.history['binary_accuracy'][49]
```

```
Matriu de confusió:
 [[250   9]
 [  6 235]]
```

```
# Codi 13: RNN

from tensorflow.keras.layers import LSTM, Dense

model = Sequential()
model.add(LSTM(32, input_shape=(train_features.shape[1], 4)))
model.add(Dense(16, activation='relu'))
model.add(Dense(2,activation='sigmoid'))
print(model.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 32)                4736
_________________________________________________________________
dense_28 (Dense)             (None, 16)                528
_________________________________________________________________
dense_29 (Dense)             (None, 2)                 34
=================================================================
```

```
Total params: 5,298
Trainable params: 5,298
Non-trainable params: 0
_________________________________________________________________
None

# Codi 13.2

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['
binary_accuracy'])

history = model.fit(train_features, train_labels,
                    epochs=50, verbose=1, validation_split=0.25)

Train on 1125 samples, validate on 375 samples
Epoch 1/50
1125/1125 [==============================] - 3s 3ms/sample - loss: 0.6
928 - binary_accuracy: 0.5093 - val_loss: 0.6908 - val_binary_accuracy
: 0.5173
Epoch 2/50
1125/1125 [==============================] - 2s 2ms/sample - loss: 0.6
787 - binary_accuracy: 0.5876 - val_loss: 0.6144 - val_binary_accuracy
: 0.7507
Epoch 3/50
1125/1125 [==============================] - 2s 2ms/sample - loss: 0.6
078 - binary_accuracy: 0.7173 - val_loss: 0.5590 - val_binary_accuracy
: 0.7333
Epoch 4/50
1125/1125 [==============================] - 2s 2ms/sample - loss: 0.5
553 - binary_accuracy: 0.7347 - val_loss: 0.5020 - val_binary_accuracy
: 0.7707
Epoch 5/50
1125/1125 [==============================] - 2s 2ms/sample - loss: 0.5
223 - binary_accuracy: 0.7520 - val_loss: 0.4889 - val_binary_accuracy
: 0.7653
Epoch 6/50
1125/1125 [==============================] - 2s 2ms/sample - loss: 0.5
156 - binary_accuracy: 0.7551 - val_loss: 0.4583 - val_binary_accuracy
: 0.7827
Epoch 7/50
1125/1125 [==============================] - 2s 2ms/sample - loss: 0.5
238 - binary_accuracy: 0.7462 - val_loss: 0.4715 - val_binary_accuracy
: 0.7707
Epoch 8/50
1125/1125 [==============================] - 2s 2ms/sample - loss: 0.5
090 - binary_accuracy: 0.7613 - val_loss: 0.4543 - val_binary_accuracy
: 0.8227
Epoch 9/50
1125/1125 [==============================] - 2s 2ms/sample - loss: 0.4
998 - binary_accuracy: 0.7751 - val_loss: 0.4482 - val_binary_accuracy
: 0.8013
Epoch 10/50
1125/1125 [==============================] - 2s 2ms/sample - loss: 0.4
783 - binary_accuracy: 0.7760 - val_loss: 0.4472 - val_binary_accuracy
```

```
: 0.7693
Epoch 11/50
1125/1125 [==============================] - 2s 2ms/sample - loss: 0.4
833 - binary_accuracy: 0.7764 - val_loss: 0.4776 - val_binary_accuracy
: 0.7693
Epoch 12/50
1125/1125 [==============================] - 2s 2ms/sample - loss: 0.4
592 - binary_accuracy: 0.7929 - val_loss: 0.4170 - val_binary_accuracy
: 0.8040
Epoch 13/50
1125/1125 [==============================] - 2s 2ms/sample - loss: 0.4
538 - binary_accuracy: 0.7836 - val_loss: 0.3925 - val_binary_accuracy
: 0.8400
Epoch 14/50
1125/1125 [==============================] - 2s 2ms/sample - loss: 0.4
339 - binary_accuracy: 0.8089 - val_loss: 0.4060 - val_binary_accuracy
: 0.8200
Epoch 15/50
1125/1125 [==============================] - 2s 2ms/sample - loss: 0.4
450 - binary_accuracy: 0.7991 - val_loss: 0.3991 - val_binary_accuracy
: 0.8293
Epoch 16/50
1125/1125 [==============================] - 2s 2ms/sample - loss: 0.4
136 - binary_accuracy: 0.8244 - val_loss: 0.3691 - val_binary_accuracy
: 0.8440
Epoch 17/50
1125/1125 [==============================] - 2s 2ms/sample - loss: 0.4
031 - binary_accuracy: 0.8218 - val_loss: 0.4013 - val_binary_accuracy
: 0.8307
Epoch 18/50
1125/1125 [==============================] - 2s 2ms/sample - loss: 0.3
812 - binary_accuracy: 0.8413 - val_loss: 0.3463 - val_binary_accuracy
: 0.8587
Epoch 19/50
1125/1125 [==============================] - 2s 2ms/sample - loss: 0.3
942 - binary_accuracy: 0.8311 - val_loss: 0.3328 - val_binary_accuracy
: 0.8667
Epoch 20/50
1125/1125 [==============================] - 2s 2ms/sample - loss: 0.3
927 - binary_accuracy: 0.8280 - val_loss: 0.3165 - val_binary_accuracy
: 0.8760
Epoch 21/50
1125/1125 [==============================] - 2s 2ms/sample - loss: 0.3
289 - binary_accuracy: 0.8631 - val_loss: 0.3690 - val_binary_accuracy
: 0.8520
Epoch 22/50
1125/1125 [==============================] - 2s 2ms/sample - loss: 0.2
899 - binary_accuracy: 0.8924 - val_loss: 0.3008 - val_binary_accuracy
: 0.8667
Epoch 23/50
1125/1125 [==============================] - 2s 2ms/sample - loss: 0.3
147 - binary_accuracy: 0.8684 - val_loss: 0.3401 - val_binary_accuracy
: 0.8440
Epoch 24/50
```

```
1125/1125 [==============================] - 2s 2ms/sample - loss: 0.2
588 - binary_accuracy: 0.8951 - val_loss: 0.2712 - val_binary_accuracy
: 0.8947
Epoch 25/50
1125/1125 [==============================] - 2s 2ms/sample - loss: 0.2
313 - binary_accuracy: 0.9156 - val_loss: 0.3436 - val_binary_accuracy
: 0.8653
Epoch 26/50
1125/1125 [==============================] - 2s 2ms/sample - loss: 0.2
118 - binary_accuracy: 0.9240 - val_loss: 0.2185 - val_binary_accuracy
: 0.9347
Epoch 27/50
1125/1125 [==============================] - 2s 2ms/sample - loss: 0.2
261 - binary_accuracy: 0.9200 - val_loss: 0.1903 - val_binary_accuracy
: 0.9267
Epoch 28/50
1125/1125 [==============================] - 2s 2ms/sample - loss: 0.1
767 - binary_accuracy: 0.9418 - val_loss: 0.1963 - val_binary_accuracy
: 0.9427
Epoch 29/50
1125/1125 [==============================] - 2s 2ms/sample - loss: 0.1
423 - binary_accuracy: 0.9587 - val_loss: 0.2199 - val_binary_accuracy
: 0.9227
Epoch 30/50
1125/1125 [==============================] - 2s 2ms/sample - loss: 0.1
692 - binary_accuracy: 0.9502 - val_loss: 0.1757 - val_binary_accuracy
: 0.9413
Epoch 31/50
1125/1125 [==============================] - 2s 2ms/sample - loss: 0.1
795 - binary_accuracy: 0.9413 - val_loss: 0.2046 - val_binary_accuracy
: 0.9280
Epoch 32/50
1125/1125 [==============================] - 2s 2ms/sample - loss: 0.1
276 - binary_accuracy: 0.9604 - val_loss: 0.1250 - val_binary_accuracy
: 0.9693
Epoch 33/50
1125/1125 [==============================] - 2s 2ms/sample - loss: 0.1
145 - binary_accuracy: 0.9698 - val_loss: 0.1613 - val_binary_accuracy
: 0.9467
Epoch 34/50
1125/1125 [==============================] - 2s 2ms/sample - loss: 0.1
317 - binary_accuracy: 0.9640 - val_loss: 0.1188 - val_binary_accuracy
: 0.9627
Epoch 35/50
1125/1125 [==============================] - 2s 2ms/sample - loss: 0.0
919 - binary_accuracy: 0.9778 - val_loss: 0.0797 - val_binary_accuracy
: 0.9787
Epoch 36/50
1125/1125 [==============================] - 2s 2ms/sample - loss: 0.0
976 - binary_accuracy: 0.9698 - val_loss: 0.1406 - val_binary_accuracy
: 0.9600
Epoch 37/50
1125/1125 [==============================] - 2s 2ms/sample - loss: 0.0
977 - binary_accuracy: 0.9729 - val_loss: 0.0779 - val_binary_accuracy
```

```
: 0.9760
Epoch 38/50
1125/1125 [==============================] - 2s 2ms/sample - loss: 0.0
750 - binary_accuracy: 0.9822 - val_loss: 0.0836 - val_binary_accuracy
: 0.9813
Epoch 39/50
1125/1125 [==============================] - 2s 2ms/sample - loss: 0.0
553 - binary_accuracy: 0.9858 - val_loss: 0.0581 - val_binary_accuracy
: 0.9800
Epoch 40/50
1125/1125 [==============================] - 2s 2ms/sample - loss: 0.0
527 - binary_accuracy: 0.9840 - val_loss: 0.0745 - val_binary_accuracy
: 0.9813
Epoch 41/50
1125/1125 [==============================] - 3s 2ms/sample - loss: 0.0
369 - binary_accuracy: 0.9911 - val_loss: 0.0559 - val_binary_accuracy
: 0.9840
Epoch 42/50
1125/1125 [==============================] - 2s 2ms/sample - loss: 0.0
642 - binary_accuracy: 0.9813 - val_loss: 0.1080 - val_binary_accuracy
: 0.9747
Epoch 43/50
1125/1125 [==============================] - 2s 2ms/sample - loss: 0.0
775 - binary_accuracy: 0.9791 - val_loss: 0.0533 - val_binary_accuracy
: 0.9840
Epoch 44/50
1125/1125 [==============================] - 2s 2ms/sample - loss: 0.0
442 - binary_accuracy: 0.9902 - val_loss: 0.0475 - val_binary_accuracy
: 0.9867
Epoch 45/50
1125/1125 [==============================] - 3s 2ms/sample - loss: 0.0
236 - binary_accuracy: 0.9947 - val_loss: 0.0382 - val_binary_accuracy
: 0.9893
Epoch 46/50
1125/1125 [==============================] - 2s 2ms/sample - loss: 0.0
160 - binary_accuracy: 0.9973 - val_loss: 0.0373 - val_binary_accuracy
: 0.9893
Epoch 47/50
1125/1125 [==============================] - 3s 3ms/sample - loss: 0.0
135 - binary_accuracy: 0.9973 - val_loss: 0.0315 - val_binary_accuracy
: 0.9893
Epoch 48/50
1125/1125 [==============================] - 2s 2ms/sample - loss: 0.0
109 - binary_accuracy: 0.9991 - val_loss: 0.0457 - val_binary_accuracy
: 0.9893
Epoch 49/50
1125/1125 [==============================] - 2s 2ms/sample - loss: 0.0
262 - binary_accuracy: 0.9929 - val_loss: 0.0850 - val_binary_accuracy
: 0.9813
Epoch 50/50
1125/1125 [==============================] - 2s 2ms/sample - loss: 0.0
659 - binary_accuracy: 0.9831 - val_loss: 0.1158 - val_binary_accuracy
: 0.9680
```

```
# Codi 13.3

predicted_labels = model.predict(np.stack(test_features))
cm = confusion_matrix(np.argmax(test_labels, axis=1),
                      np.argmax(predicted_labels, axis=1))
print('Matriu de confusió:\n',cm)
```

```
Matriu de confusió:
 [[254   5]
 [  6 235]]
```

9.2. Annex 2

```
# Codi 1

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os

print(os.listdir("../Problema 2/asl-alphabet/"))
train_dir="../Problema 2/asl-alphabet/asl_alphabet_train/"
test_dir="../Problema 2/asl-alphabet/asl_alphabet_test/"

import keras
from keras.layers import Conv2D,Dense,Flatten,MaxPooling2D,Dropout
from keras.models import Sequential
from keras.preprocessing.image import ImageDataGenerator
```

```
['.DS_Store', 'asl_alphabet_test', 'asl_alphabet_train']
```

```
Using TensorFlow backend.
```

Construcció del model

```
# Codi 2

model=Sequential()
model.add(Conv2D(32,kernel_size=(3,3),input_shape=(64,64,3),activation
='relu',padding='same'))
model.add(Conv2D(32,kernel_size=(3,3),activation='relu',padding='same'
))
model.add(MaxPooling2D(pool_size=(3,3)))
model.add(Conv2D(64,kernel_size=(3,3),activation='relu',padding='same'
))
model.add(Conv2D(64,kernel_size=(3,3),activation='relu',padding='same'
))
model.add(MaxPooling2D(pool_size=(3,3)))
model.add(Conv2D(128,kernel_size=(3,3),activation='relu',padding='same
'))
model.add(Conv2D(128,kernel_size=(3,3),activation='relu',padding='same
'))
model.add(MaxPooling2D(pool_size=(3,3)))
model.add(Flatten())
model.add(Dropout(0.45))
model.add(Dense(512,activation='relu'))
model.add(Dense(29,activation='softmax'))

model.summary()
model.compile(loss='categorical_crossentropy',metrics=['accuracy'],opt
imizer='adam')
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 64, 64, 32)        896
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 64, 64, 32)        9248
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 21, 21, 32)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 21, 21, 64)        18496
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 21, 21, 64)        36928
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 7, 7, 128)         73856
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 7, 7, 128)         147584
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 2, 2, 128)         0
_________________________________________________________________
flatten_2 (Flatten)          (None, 512)               0
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_3 (Dense)              (None, 512)               262656
_________________________________________________________________
dense_4 (Dense)              (None, 29)                14877
=================================================================
Total params: 564,541
Trainable params: 564,541
Non-trainable params: 0
_________________________________________________________________
```

```
# Codi 3

train_data_gen=ImageDataGenerator(rescale=1./255,
                                  validation_split=0.2)

test_data_gen=ImageDataGenerator(rescale=1/.255)

train_generator=train_data_gen.flow_from_directory(train_dir,
                                                   target_size=(64,64),
                                                   batch_size=32,
                                                   class_mode='categorical
', subset='training')

validation_generator=train_data_gen.flow_from_directory(train_dir,
                                                   target_size=(64,64),
                                                   batch_size=32,
                                                   class_mode='categorical
',subset='validation',
shuffle=False)
```

```
test_generator=test_data_gen.flow_from_directory(test_dir,
                                                 target_size=(64,64),
                                                 batch_size=1,
                                                 class_mode='categorica
l', shuffle=False)
```

```
Found 69600 images belonging to 29 classes.
Found 17400 images belonging to 29 classes.
Found 29 images belonging to 29 classes.
```

```
# Codi 4

callbacks=[
    keras.callbacks.ReduceLROnPlateau(
        monitor='val_loss',
        factor=0.1,
        patience=10),
    keras.callbacks.EarlyStopping(
        monitor='acc',
        patience=4)
]
```

```
# Codi 5

history=model.fit_generator(train_generator,
                            steps_per_epoch=90,
                            epochs=30,
                            validation_data=validation_generator,
                            validation_steps=15,
                            callbacks=callbacks)
```

```
Epoch 1/30
90/90 [==============================] - 48s 536ms/step - loss: 3.3709
- acc: 0.0312 - val_loss: 3.3651 - val_acc: 0.0271
Epoch 2/30
90/90 [==============================] - 49s 540ms/step - loss: 3.3526
- acc: 0.0431 - val_loss: 3.3445 - val_acc: 0.0583
Epoch 3/30
90/90 [==============================] - 46s 507ms/step - loss: 3.0933
- acc: 0.1122 - val_loss: 2.7185 - val_acc: 0.2292
Epoch 4/30
90/90 [==============================] - 45s 499ms/step - loss: 2.1131
- acc: 0.3458 - val_loss: 1.8693 - val_acc: 0.4229
Epoch 5/30
90/90 [==============================] - 47s 523ms/step - loss: 1.4076
- acc: 0.5257 - val_loss: 1.5354 - val_acc: 0.5458
Epoch 6/30
90/90 [==============================] - 45s 497ms/step - loss: 1.0561
- acc: 0.6476 - val_loss: 1.1802 - val_acc: 0.5958
Epoch 7/30
90/90 [==============================] - 46s 511ms/step - loss: 0.7918
- acc: 0.7257 - val_loss: 1.0763 - val_acc: 0.6708
Epoch 8/30
90/90 [==============================] - 45s 501ms/step - loss: 0.6025
- acc: 0.7903 - val_loss: 1.0045 - val_acc: 0.6833
```

```
Epoch 9/30
90/90 [==============================] - 46s 507ms/step - loss: 0.5302
- acc: 0.8215 - val_loss: 0.7887 - val_acc: 0.7583
Epoch 10/30
90/90 [==============================] - 46s 508ms/step - loss: 0.4401
- acc: 0.8444 - val_loss: 0.8223 - val_acc: 0.7375
Epoch 11/30
90/90 [==============================] - 45s 504ms/step - loss: 0.3997
- acc: 0.8514 - val_loss: 0.7513 - val_acc: 0.7583
Epoch 12/30
90/90 [==============================] - 46s 511ms/step - loss: 0.3311
- acc: 0.8743 - val_loss: 0.5788 - val_acc: 0.8271
Epoch 13/30
90/90 [==============================] - 45s 506ms/step - loss: 0.3296
- acc: 0.8854 - val_loss: 0.6118 - val_acc: 0.8000
Epoch 14/30
90/90 [==============================] - 46s 508ms/step - loss: 0.2611
- acc: 0.9038 - val_loss: 0.6062 - val_acc: 0.8146
Epoch 15/30
90/90 [==============================] - 45s 503ms/step - loss: 0.2671
- acc: 0.9049 - val_loss: 0.5908 - val_acc: 0.8042
Epoch 16/30
90/90 [==============================] - 45s 502ms/step - loss: 0.2612
- acc: 0.9118 - val_loss: 0.5704 - val_acc: 0.8688
Epoch 17/30
90/90 [==============================] - 46s 509ms/step - loss: 0.2265
- acc: 0.9177 - val_loss: 0.5016 - val_acc: 0.8354
Epoch 18/30
90/90 [==============================] - 45s 501ms/step - loss: 0.2309
- acc: 0.9167 - val_loss: 0.5078 - val_acc: 0.8750
Epoch 19/30
90/90 [==============================] - 45s 500ms/step - loss: 0.1663
- acc: 0.9382 - val_loss: 0.7477 - val_acc: 0.8000
Epoch 20/30
90/90 [==============================] - 45s 496ms/step - loss: 0.2119
- acc: 0.9243 - val_loss: 0.4421 - val_acc: 0.8646
Epoch 21/30
90/90 [==============================] - 45s 502ms/step - loss: 0.1860
- acc: 0.9392 - val_loss: 0.4221 - val_acc: 0.8812
Epoch 22/30
90/90 [==============================] - 45s 501ms/step - loss: 0.2058
- acc: 0.9333 - val_loss: 0.5410 - val_acc: 0.8333
Epoch 23/30
90/90 [==============================] - 45s 502ms/step - loss: 0.1556
- acc: 0.9451 - val_loss: 0.5991 - val_acc: 0.8521
Epoch 24/30
90/90 [==============================] - 45s 504ms/step - loss: 0.1671
- acc: 0.9517 - val_loss: 0.4079 - val_acc: 0.8833
Epoch 25/30
90/90 [==============================] - 46s 511ms/step - loss: 0.1425
- acc: 0.9500 - val_loss: 0.5365 - val_acc: 0.8521
Epoch 26/30
90/90 [==============================] - 45s 501ms/step - loss: 0.1699
- acc: 0.9424 - val_loss: 0.6406 - val_acc: 0.8167
```

```
Epoch 27/30
90/90 [==============================] - 46s 510ms/step - loss: 0.1285
- acc: 0.9594 - val_loss: 0.5307 - val_acc: 0.8500
Epoch 28/30
90/90 [==============================] - 45s 504ms/step - loss: 0.1233
- acc: 0.9535 - val_loss: 0.4867 - val_acc: 0.8792
Epoch 29/30
90/90 [==============================] - 45s 504ms/step - loss: 0.1253
- acc: 0.9590 - val_loss: 0.3588 - val_acc: 0.9042
Epoch 30/30
90/90 [==============================] - 45s 505ms/step - loss: 0.1260
- acc: 0.9566 - val_loss: 0.5230 - val_acc: 0.8792
```

Avaluació

```
# Codi 6

model.save("ASL4.h5")
hist=history.history
print(hist.keys()
     )

val_acc=hist['val_acc']
val_loss=hist['val_loss']
acc=hist['acc']
loss=hist['loss']

import matplotlib.pyplot as plt
epochs=range(1,len(acc)+1)

print(epochs)
```

```
dict_keys(['val_loss', 'val_acc', 'loss', 'acc', 'lr'])
range(1, 31)
```

```
# Per carregar el problema de nou
from keras.models import load_model
model = load_model('ASL4.h5')

# Codi 7

# Gràfic accuracy
plt.plot(epochs,acc,'#d9223a',label='Training_acc')
plt.plot(epochs,val_acc,'#595959',label='Validation_acc')
plt.title('Training_vs Validation Accuracy')
plt.legend()
plt.figure()

# Gràfic loss
plt.plot(epochs,loss,'#d9223a',label='Training_loss')
plt.plot(epochs,val_loss,'#595959',label='Validation_loss')
plt.title('Training_vs Validation Loss')
```

```
plt.legend()
plt.figure()
```

```
<Figure size 432x288 with 0 Axes>
```

```
# Codi 8: Evaluate amb validation_gene

evaluate=model.evaluate_generator(validation_generator, steps=544)
print("Loss:",evaluate[0])
print("Acc:",evaluate[1])
```

```
Loss: 0.4811159350924729
Acc: 0.8778199815837937
```

```
# Codi 8: Predict amb validation_gene

from sklearn.metrics import classification_report, confusion_matrix

Y_pred = model.predict_generator(validation_generator, steps=544)
y_pred = np.argmax(Y_pred, axis=1)

# Codi 8 validation report
target_names = ['A','B','C','D','Del','E','F','G','H','I','J','K','L',
'M','N',
                'Nothing','O','P','K','R','S','Space','R','U','V','W',
'X','Y','Z']
print(classification_report(validation_generator.classes, y_pred, targ
et_names=target_names))
```

```
              precision    recall  f1-score   support

           A       0.81      0.96      0.88       600
           B       1.00      0.74      0.85       600
           C       0.93      1.00      0.96       600
           D       0.98      0.99      0.99       600
         Del       0.72      0.95      0.82       600
           E       0.96      0.95      0.95       600
           F       0.96      0.50      0.66       600
           G       0.61      1.00      0.76       600
           H       1.00      0.74      0.85       600
           I       0.82      0.89      0.85       600
           J       0.97      0.96      0.96       600
           K       1.00      1.00      1.00       600
           L       0.63      0.81      0.71       600
           M       0.69      0.49      0.57       600
           N       0.93      0.79      0.85       600
     Nothing       0.98      0.95      0.97       600
           O       0.97      1.00      0.99       600
           P       0.93      0.95      0.94       600
           K       0.65      0.82      0.73       600
           R       0.98      0.76      0.85       600
           S       0.91      0.96      0.94       600
       Space       0.92      0.93      0.93       600
           R       0.89      0.94      0.92       600
           U       0.96      0.56      0.71       600
```

```
           V       0.91      0.93      0.92       600
           W       0.88      0.97      0.92       600
           X       0.98      0.96      0.97       600
           Y       0.95      0.96      0.96       600
           Z       0.99      1.00      0.99       600

   micro avg       0.88      0.88      0.88     17400
   macro avg       0.89      0.88      0.88     17400
weighted avg       0.89      0.88      0.88     17400
```

```
Y_pred = model.predict_generator(test_generator, steps=29)
y_pred = np.argmax(Y_pred, axis=1)

target_names = ['A','B','C','D','Del','E','F','G','H','I','J','K','L',
'M','N',
                'Nothing','O','P','K','R','S','Space','R','U','V','W',
'X','Y','Z']
print(classification_report(test_generator.classes, y_pred, target_nam
es=target_names))
```

```
              precision    recall  f1-score   support

           A       0.00      0.00      0.00         1
           B       0.00      0.00      0.00         1
           C       0.00      0.00      0.00         1
           D       0.00      0.00      0.00         1
         Del       0.00      0.00      0.00         1
           E       0.00      0.00      0.00         1
           F       0.00      0.00      0.00         1
           G       0.00      0.00      0.00         1
           H       0.00      0.00      0.00         1
           I       0.00      0.00      0.00         1
           J       0.00      0.00      0.00         1
           K       0.00      0.00      0.00         1
           L       0.00      0.00      0.00         1
           M       0.00      0.00      0.00         1
           N       0.00      0.00      0.00         1
     Nothing       0.00      0.00      0.00         1
           O       0.00      0.00      0.00         1
           P       0.00      0.00      0.00         1
           K       0.00      0.00      0.00         1
           R       0.00      0.00      0.00         1
           S       0.00      0.00      0.00         1
       Space       0.00      0.00      0.00         1
           R       0.00      0.00      0.00         1
           U       0.00      0.00      0.00         1
           V       0.00      0.00      0.00         1
           W       0.00      0.00      0.00         1
           X       0.00      0.00      0.00         1
           Y       0.00      0.00      0.00         1
           Z       0.00      0.00      0.00         1

   micro avg       0.00      0.00      0.00        29
```

```
   macro avg       0.00      0.00      0.00        29
weighted avg       0.00      0.00      0.00        29
```

9.3. Annex 3

Importació paquets

```
# Codi 1

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os

print(os.listdir("../Problema 2.2/asl-alphabet/"))
train_dir="../Problema 2.2/asl-alphabet/asl_alphabet_train/"
test_dir="../Problema 2.2/asl-alphabet/asl_alphabet_test/"

import keras
from keras.layers import Conv2D,Dense,Flatten,MaxPooling2D,Dropout
from keras.models import Sequential
from keras.preprocessing.image import ImageDataGenerator
```

```
['.DS_Store', 'asl_alphabet_test', 'asl_alphabet_train']
```

```
Using TensorFlow backend.
```

Definició del model

```
# Codi 2

model=Sequential()
model.add(Conv2D(32,kernel_size=(3,3),input_shape=(64,64,3),activation
='relu',padding='same'))
model.add(Conv2D(32,kernel_size=(3,3),activation='relu',padding='same'
))
model.add(MaxPooling2D(pool_size=(3,3)))
model.add(Conv2D(64,kernel_size=(3,3),activation='relu',padding='same'
))
model.add(Conv2D(64,kernel_size=(3,3),activation='relu',padding='same'
))
model.add(MaxPooling2D(pool_size=(3,3)))
model.add(Conv2D(128,kernel_size=(3,3),activation='relu',padding='same
'))
model.add(Conv2D(128,kernel_size=(3,3),activation='relu',padding='same
'))
model.add(MaxPooling2D(pool_size=(3,3)))
model.add(Flatten())
model.add(Dropout(0.45))
model.add(Dense(512,activation='relu'))
model.add(Dense(29,activation='softmax'))

model.summary()
model.compile(loss='categorical_crossentropy',metrics=['accuracy'],opt
imizer='adam')
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_7 (Conv2D)            (None, 64, 64, 32)        896
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 64, 64, 32)        9248
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 21, 21, 32)        0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 21, 21, 64)        18496
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 21, 21, 64)        36928
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 7, 7, 128)         73856
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 7, 7, 128)         147584
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 2, 2, 128)         0
_________________________________________________________________
flatten_2 (Flatten)          (None, 512)               0
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_3 (Dense)              (None, 512)               262656
_________________________________________________________________
dense_4 (Dense)              (None, 29)                14877
=================================================================
Total params: 564,541
Trainable params: 564,541
Non-trainable params: 0
_________________________________________________________________
```

Tractament de les dades

```
# Codi 3

train_data_gen=ImageDataGenerator(rescale=1./255,
                                  validation_split=0.2)

test_data_gen=ImageDataGenerator(rescale=1/.255)

train_generator=train_data_gen.flow_from_directory(train_dir,
                                                   target_size=(64,64),
                                                   batch_size=32,
                                                   class_mode='categorical
', subset='training')

validation_generator=train_data_gen.flow_from_directory(train_dir,
                                                   target_size=(64,64),
                                                   batch_size=32,
```

```
                                                    class_mode='categorical
', subset='validation', shuffle=False)

test_generator=test_data_gen.flow_from_directory(test_dir,
                                                  target_size=(64,64),
                                                  batch_size=32,
                                                  class_mode='categorica
l', shuffle=False)
```

```
Found 64960 images belonging to 29 classes.
Found 16240 images belonging to 29 classes.
Found 5829 images belonging to 29 classes.
```

Ajust del model

```
# Codi 4

callbacks=[
    keras.callbacks.ReduceLROnPlateau(
        monitor='val_loss',
        factor=0.1,
        patience=10),
    keras.callbacks.EarlyStopping(
        monitor='acc',
        patience=4)
]

# Codi 5

history=model.fit_generator(train_generator,
                            steps_per_epoch=90,
                            epochs=30,
                            validation_data=validation_generator,
                            validation_steps=15,
                            callbacks=callbacks)
```

```
WARNING:tensorflow:From /anaconda3/lib/python3.7/site-packages/tensorf
low/python/ops/math_ops.py:3066: to_int32 (from tensorflow.python.ops.
math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Epoch 1/30
90/90 [==============================] - 48s 533ms/step - loss: 3.3682
- acc: 0.0337 - val_loss: 3.3579 - val_acc: 0.0000e+00
Epoch 2/30
90/90 [==============================] - 46s 511ms/step - loss: 3.3676
- acc: 0.0358 - val_loss: 3.3912 - val_acc: 0.0000e+00
Epoch 3/30
90/90 [==============================] - 45s 500ms/step - loss: 3.3016
- acc: 0.0528 - val_loss: 2.2119 - val_acc: 0.4917
Epoch 4/30
90/90 [==============================] - 47s 517ms/step - loss: 2.9139
- acc: 0.1219 - val_loss: 1.8493 - val_acc: 0.3646
Epoch 5/30
90/90 [==============================] - 48s 532ms/step - loss: 2.3632
```

```
- acc: 0.2563 - val_loss: 1.5426 - val_acc: 0.4417
Epoch 6/30
90/90 [==============================] - 47s 522ms/step - loss: 1.8054
- acc: 0.4062 - val_loss: 2.0510 - val_acc: 0.1854
Epoch 7/30
90/90 [==============================] - 47s 518ms/step - loss: 1.3762
- acc: 0.5406 - val_loss: 0.2961 - val_acc: 0.9271
Epoch 8/30
90/90 [==============================] - 49s 543ms/step - loss: 1.0536
- acc: 0.6306 - val_loss: 0.6412 - val_acc: 0.8250
Epoch 9/30
90/90 [==============================] - 47s 522ms/step - loss: 0.7886
- acc: 0.7250 - val_loss: 0.2770 - val_acc: 0.8958
Epoch 10/30
90/90 [==============================] - 47s 523ms/step - loss: 0.6415
- acc: 0.7722 - val_loss: 2.3141 - val_acc: 0.5625
Epoch 11/30
90/90 [==============================] - 59s 656ms/step - loss: 0.5484
- acc: 0.8031 - val_loss: 2.1972 - val_acc: 0.3146
Epoch 12/30
90/90 [==============================] - 53s 591ms/step - loss: 0.4383
- acc: 0.8469 - val_loss: 1.0431 - val_acc: 0.5813
Epoch 13/30
90/90 [==============================] - 53s 594ms/step - loss: 0.4066
- acc: 0.8576 - val_loss: 0.3292 - val_acc: 0.8979
Epoch 14/30
90/90 [==============================] - 54s 595ms/step - loss: 0.3548
- acc: 0.8747 - val_loss: 0.4210 - val_acc: 0.8917
Epoch 15/30
90/90 [==============================] - 51s 570ms/step - loss: 0.2869
- acc: 0.8986 - val_loss: 1.1616 - val_acc: 0.5521
Epoch 16/30
90/90 [==============================] - 55s 615ms/step - loss: 0.2825
- acc: 0.9056 - val_loss: 3.7263 - val_acc: 0.1938
Epoch 17/30
90/90 [==============================] - 56s 620ms/step - loss: 0.2472
- acc: 0.9146 - val_loss: 0.6667 - val_acc: 0.7708
Epoch 18/30
90/90 [==============================] - 49s 542ms/step - loss: 0.2384
- acc: 0.9226 - val_loss: 0.7176 - val_acc: 0.8292
Epoch 19/30
90/90 [==============================] - 51s 570ms/step - loss: 0.2109
- acc: 0.9295 - val_loss: 0.2284 - val_acc: 0.9229
Epoch 20/30
90/90 [==============================] - 53s 586ms/step - loss: 0.2114
- acc: 0.9285 - val_loss: 0.1273 - val_acc: 0.9521
Epoch 21/30
90/90 [==============================] - 54s 599ms/step - loss: 0.1946
- acc: 0.9375 - val_loss: 0.4637 - val_acc: 0.8146
Epoch 22/30
90/90 [==============================] - 55s 614ms/step - loss: 0.1846
- acc: 0.9358 - val_loss: 0.3181 - val_acc: 0.9479
Epoch 23/30
90/90 [==============================] - 51s 566ms/step - loss: 0.1669
```

```
- acc: 0.9462 - val_loss: 0.6218 - val_acc: 0.8354
Epoch 24/30
90/90 [==============================] - 46s 506ms/step - loss: 0.1751
- acc: 0.9431 - val_loss: 0.6477 - val_acc: 0.7438
Epoch 25/30
90/90 [==============================] - 45s 500ms/step - loss: 0.1190
- acc: 0.9604 - val_loss: 0.2207 - val_acc: 0.9396
Epoch 26/30
90/90 [==============================] - 45s 502ms/step - loss: 0.1363
- acc: 0.9528 - val_loss: 1.8139 - val_acc: 0.5687
Epoch 27/30
90/90 [==============================] - 45s 501ms/step - loss: 0.1188
- acc: 0.9615 - val_loss: 0.5442 - val_acc: 0.8750
Epoch 28/30
90/90 [==============================] - 45s 502ms/step - loss: 0.1243
- acc: 0.9576 - val_loss: 1.0983 - val_acc: 0.5896
Epoch 29/30
90/90 [==============================] - 44s 494ms/step - loss: 0.1107
- acc: 0.9611 - val_loss: 0.3982 - val_acc: 0.8896
Epoch 30/30
90/90 [==============================] - 45s 495ms/step - loss: 0.0996
- acc: 0.9688 - val_loss: 0.2716 - val_acc: 0.8812
```

```
# Codi 6

model.save("ASL10.h5")
hist=history.history
print(hist.keys())

val_acc=hist['val_acc']
val_loss=hist['val_loss']
acc=hist['acc']
loss=hist['loss']

import matplotlib.pyplot as plt
epochs=range(1,len(acc)+1)

print(epochs)
```

```
dict_keys(['val_loss', 'val_acc', 'loss', 'acc', 'lr'])
range(1, 31)
```

```
# Per carregar el problema de nou
from keras.models import load_model
model = load_model('ASL10.h5')
```

```
WARNING:tensorflow:From /anaconda3/lib/python3.7/site-packages/tensorf
low/python/framework/op_def_library.py:263: colocate_with (from tensor
flow.python.framework.ops) is deprecated and will be removed in a futu
re version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From /anaconda3/lib/python3.7/site-packages/keras/b
ackend/tensorflow_backend.py:3445: calling dropout (from tensorflow.py
thon.ops.nn_ops) with keep_prob is deprecated and will be removed in a
```

```
future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate
= 1 - keep_prob`.
WARNING:tensorflow:From /anaconda3/lib/python3.7/site-packages/tensorf
low/python/ops/math_ops.py:3066: to_int32 (from tensorflow.python.ops.
math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
```

Avaluació i predicció

```
# Codi 7

# Gràfic accuracy
plt.plot(epochs,acc,'#d9223a',label='Training_acc')
plt.plot(epochs,val_acc,'#595959',label='Validation_acc')
plt.title('Training_vs Validation Accuracy')
plt.legend()
plt.figure()

# Gràfic loss
plt.plot(epochs,loss,'#d9223a',label='Training_loss')
plt.plot(epochs,val_loss,'#595959',label='Validation_loss')
plt.title('Training_vs Validation Loss')
plt.legend()
plt.figure()
```

```
<Figure size 432x288 with 0 Axes>
```

```
<Figure size 432x288 with 0 Axes>
```

```
# Codi 8.1: Evaluate amb validation_generator

evaluate=model.evaluate_generator(validation_generator, steps=508)
print("Loss:",evaluate[0])
print("Acc:",evaluate[1])
```

```
Loss: 0.5461474592727072
Acc: 0.8269088669950739
```

```
# Codi 8.2: Predict amb validation_generator

from sklearn.metrics import classification_report

Y_pred = model.predict_generator(validation_generator, steps=508)
y_pred = np.argmax(Y_pred, axis=1)

# Codi 8.3: Validation_generator report

target_names = ['A','B','C','D','Del','E','F','G','H','I','J','K','L',
'M','N',
                'Not','O','P','Q','R','S','Space','T','U','V','W','X',
'Y','Z']
print(classification_report(validation_generator.classes, y_pred, targ
et_names=target_names))
```

```
              precision    recall  f1-score   support

           A       0.22      0.16      0.18       560
           B       0.23      0.23      0.23       560
           C       0.22      0.20      0.21       560
           D       0.29      0.23      0.26       560
         Del       0.16      0.24      0.19       560
           E       0.20      0.23      0.21       560
           F       0.16      0.15      0.15       560
           G       0.27      0.33      0.29       560
           H       0.21      0.15      0.18       560
           I       0.29      0.27      0.28       560
           J       0.22      0.23      0.22       560
           K       0.22      0.23      0.23       560
           L       0.11      0.11      0.11       560
           M       0.37      0.35      0.36       560
           N       0.19      0.20      0.19       560
         Not       0.20      0.23      0.21       560
           O       0.27      0.23      0.25       560
           P       0.23      0.23      0.23       560
           Q       0.11      0.17      0.13       560
           R       0.24      0.14      0.18       560
           S       0.15      0.20      0.17       560
       Space       0.26      0.23      0.24       560
           T       0.28      0.34      0.31       560
           U       0.16      0.09      0.12       560
           V       0.23      0.22      0.23       560
           W       0.24      0.23      0.23       560
           X       0.24      0.24      0.24       560
           Y       0.22      0.23      0.23       560
           Z       0.20      0.23      0.21       560

   micro avg       0.22      0.22      0.22     16240
   macro avg       0.22      0.22      0.22     16240
weighted avg       0.22      0.22      0.22     16240
```

```
# Codi 9.1: predict amb test_generator

from sklearn.metrics import classification_report

Y_pred_test = model.predict_generator(test_generator, steps=183)
y_pred_test = np.argmax(Y_pred_test, axis=1)

# Codi 9.2: predict amb test_generator

target_names = ['A','B','C','D','Del','E','F','G','H','I','J','K','L',
'M',
                'N','Not','O','P','Q','R','S','Space','T','U','V','W',
'X','Y','Z']
print(classification_report(test_generator.classes, y_pred_test, targe
t_names=target_names))
```

	precision	recall	f1-score	support
A	0.00	0.00	0.00	201
B	0.00	0.00	0.00	201
C	0.00	0.00	0.00	201
D	0.00	0.00	0.00	201
Del	0.22	0.51	0.31	201
E	0.01	0.01	0.01	201
F	0.00	0.00	0.00	201
G	0.00	0.00	0.00	201
H	0.06	0.09	0.08	201
I	0.00	0.00	0.00	201
J	0.00	0.00	0.00	201
K	0.10	0.49	0.17	201
L	0.00	0.00	0.00	201
M	0.00	0.00	0.00	201
N	0.00	0.00	0.00	201
Not	0.00	0.00	0.00	201
O	0.00	0.00	0.00	201
P	0.00	0.00	0.00	201
Q	0.00	0.00	0.00	201
R	0.00	0.00	0.00	201
S	0.00	0.00	0.00	201
Space	0.00	0.00	0.00	201
T	0.00	0.00	0.00	201
U	0.00	0.00	0.00	201
V	0.00	0.00	0.00	201
W	0.00	0.00	0.00	201
X	0.00	0.00	0.00	201
Y	0.00	0.00	0.00	201
Z	0.01	0.00	0.01	201
micro avg	0.04	0.04	0.04	5829
macro avg	0.01	0.04	0.02	5829
weighted avg	0.01	0.04	0.02	5829

9.4. Annex 4

Base de dades

```
# Codi 1: Càrrega de les lliberies necessàries

import numpy as np
import pandas as pd

from sklearn.feature_extraction.text import CountVectorizer
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense, Embedding, LSTM, SpatialDropout1D
from sklearn.model_selection import train_test_split
from keras.utils.np_utils import to_categorical
import re
```

```
Using TensorFlow backend.
```

```
# Codi 2:

# Es carrega la bade de dades completa i es seleccionen les variables
d'inetrès
data = pd.read_csv('Sentiment.csv')
data = data[['text','sentiment']]
```

```
# Codi 3: Tractament 'Sentiment'

data = data[data.sentiment != "Neutral"]
print(data[ data['sentiment'] == 'Positive'].size)
print(data[ data['sentiment'] == 'Negative'].size)
```

```
4472
16986
```

```
# Codi 4: Tractament 'text'

# Conservar només aquells tweets que siguin text o paraules
data['text'] = data['text'].apply(lambda x: x.lower())
data['text'] = data['text'].apply((lambda x: re.sub('[^a-zA-z0-9\s]','
',x)))

for idx,row in data.iterrows():
    row[0] = row[0].replace('rt',' ')

# Nombre màxim de paraules per tweet i definició de seqüències
max_fatures = 2000
tokenizer = Tokenizer(num_words=max_fatures, split=' ')
tokenizer.fit_on_texts(data['text'].values)
X = tokenizer.texts_to_sequences(data['text'].values)
X = pad_sequences(X)
```

```
# Codi 5:

Y = pd.get_dummies(data['sentiment']).values
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0
.33, random_state = 42)
print(X_train.shape,Y_train.shape)
print(X_test.shape,Y_test.shape)
```

```
(7188, 28) (7188, 2)
(3541, 28) (3541, 2)
```

Definició del model

```
# Codi 6:

embed_dim = 128
lstm_out = 196

model = Sequential()
model.add(Embedding(input_dim=max_fatures,
                    output_dim=embed_dim,
                    input_length = X.shape[1]))
model.add(SpatialDropout1D(0.4))
model.add(LSTM(units=lstm_out, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(2,activation='softmax'))

model.compile(loss = 'categorical_crossentropy',
              optimizer='adam',
              metrics = ['accuracy'])
print(model.summary())
```

```
WARNING:tensorflow:From /anaconda3/lib/python3.7/site-packages/tensorf
low/python/framework/op_def_library.py:263: colocate_with (from tensor
flow.python.framework.ops) is deprecated and will be removed in a futu
re version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From /anaconda3/lib/python3.7/site-packages/keras/b
ackend/tensorflow_backend.py:3445: calling dropout (from tensorflow.py
thon.ops.nn_ops) with keep_prob is deprecated and will be removed in a
future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate
= 1 - keep_prob`.
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 28, 128)           256000
_________________________________________________________________
spatial_dropout1d_1 (Spatial (None, 28, 128)           0
_________________________________________________________________
lstm_1 (LSTM)                (None, 196)               254800
_________________________________________________________________
```

```
dense_1 (Dense)              (None, 2)                 394
=================================================================
Total params: 511,194
Trainable params: 511,194
Non-trainable params: 0
_________________________________________________________________
None
```

Entrenament del model

```
# Codi 7:

batch_size = 32
history=model.fit(X_train, Y_train,
                  epochs = 7,
                  batch_size=batch_size,
                  verbose = 2)
```

```
WARNING:tensorflow:From /anaconda3/lib/python3.7/site-packages/tensorf
low/python/ops/math_ops.py:3066: to_int32 (from tensorflow.python.ops.
math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Epoch 1/7
 - 15s - loss: 0.4465 - acc: 0.8146
Epoch 2/7
 - 18s - loss: 0.3251 - acc: 0.8617
Epoch 3/7
 - 14s - loss: 0.2856 - acc: 0.8816
Epoch 4/7
 - 14s - loss: 0.2530 - acc: 0.8977
Epoch 5/7
 - 14s - loss: 0.2268 - acc: 0.9086
Epoch 6/7
 - 14s - loss: 0.2120 - acc: 0.9122
Epoch 7/7
 - 14s - loss: 0.1865 - acc: 0.9222
```

```
# Codi 8:

# Per carregar el problema de nou
from keras.models import load_model
model = load_model('twt1.h5')

# Codi 9: Reprsentació model

model.save("twt1.h5")
hist=history.history
acc=hist['acc']
loss=hist['loss']

import matplotlib.pyplot as plt
epochs=range(1,len(acc)+1)

# Gràfic accuracy
```

```
plt.plot(epochs,acc,'#d9223a',label='Training_acc')
plt.title('Accuracy')
plt.legend()
plt.figure()

# Gràfic loss
plt.plot(epochs,loss,'#d9223a',label='Training_loss')
plt.title('Loss')
plt.legend()
plt.figure()

dict_keys(['loss', 'acc'])
range(1, 8)
```

Avaluació del model

```
# Codi 10: Avaluació amb les dades de validació

validation_size = 1500

X_validate = X_test[-validation_size:]
Y_validate = Y_test[-validation_size:]
X_test = X_test[:-validation_size]
Y_test = Y_test[:-validation_size]
score,acc = model.evaluate(X_test, Y_test, verbose = 2, batch_size = b
atch_size)
print("loss: %.2f" % (score))
print("acc: %.2f" % (acc))

loss: 0.50
acc: 0.82

# Codi 11:

from sklearn.metrics import confusion_matrix
import itertools

predicted_labels = model.predict(np.stack(X_test))

cm = confusion_matrix(np.argmax(Y_test, axis=1),
                      np.argmax(predicted_labels, axis=1))
print('Matriu de confusió:\n',cm)

cm = cm.astype('float') / cm.sum(axis = 1)[:, np.newaxis]

plt.imshow(cm, cmap=plt.cm.Blues)
plt.title('Matriu de confusió normalitzada')
plt.colorbar()
plt.xlabel('Nivells reals')
plt.ylabel('Nivells predits')
plt.xticks([0, 1]); plt.yticks([0, 1])
plt.grid('off')
for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
    plt.text(j, i, format(cm[i, j], '.2f'),
```

```
            horizontalalignment='center',
            color='white' if cm[i, j] > 0.5 else 'black')
```

```
Matriu de confusió:
 [[2581  240]
 [ 327  393]]
```

Modificació 1

```
# Codi 12:

# Definició model
embed_dim = 128
lstm_out = 196
model = Sequential()
model.add(Embedding(input_dim=max_fatures,
                    output_dim=embed_dim,
                    input_length = X.shape[1]))
model.add(SpatialDropout1D(0.4))
model.add(LSTM(units=lstm_out, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(2,activation='sigmoid'))
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['binary_accuracy'])

# Codi 13:

batch_size = 32
history=model.fit(X_train, Y_train,
                  epochs = 30,
                  batch_size=batch_size,
                  verbose = 2)
```

```
WARNING:tensorflow:From /anaconda3/lib/python3.7/site-packages/tensorf
low/python/ops/math_ops.py:3066: to_int32 (from tensorflow.python.ops.
math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Epoch 1/30
 - 15s - loss: 0.4575 - binary_accuracy: 0.8077
Epoch 2/30
 - 14s - loss: 0.3285 - binary_accuracy: 0.8625
Epoch 3/30
 - 25s - loss: 0.2859 - binary_accuracy: 0.8815
Epoch 4/30
 - 24s - loss: 0.2635 - binary_accuracy: 0.8950
Epoch 5/30
 - 15s - loss: 0.2335 - binary_accuracy: 0.9037
Epoch 6/30
 - 19s - loss: 0.2176 - binary_accuracy: 0.9140
Epoch 7/30
 - 18s - loss: 0.1983 - binary_accuracy: 0.9213
Epoch 8/30
```

```
 - 19s - loss: 0.1800 - binary_accuracy: 0.9281
Epoch 9/30
 - 16s - loss: 0.1649 - binary_accuracy: 0.9374
Epoch 10/30
 - 16s - loss: 0.1489 - binary_accuracy: 0.9413
Epoch 11/30
 - 15s - loss: 0.1457 - binary_accuracy: 0.9421
Epoch 12/30
 - 16s - loss: 0.1336 - binary_accuracy: 0.9440
Epoch 13/30
 - 16s - loss: 0.1262 - binary_accuracy: 0.9462
Epoch 14/30
 - 15s - loss: 0.1162 - binary_accuracy: 0.9503
Epoch 15/30
 - 18s - loss: 0.1131 - binary_accuracy: 0.9544
Epoch 16/30
 - 17s - loss: 0.1081 - binary_accuracy: 0.9566
Epoch 17/30
 - 16s - loss: 0.1047 - binary_accuracy: 0.9563
Epoch 18/30
 - 16s - loss: 0.1017 - binary_accuracy: 0.9572
Epoch 19/30
 - 18s - loss: 0.0967 - binary_accuracy: 0.9605
Epoch 20/30
 - 18s - loss: 0.0874 - binary_accuracy: 0.9644
Epoch 21/30
 - 17s - loss: 0.0872 - binary_accuracy: 0.9635
Epoch 22/30
 - 19s - loss: 0.0864 - binary_accuracy: 0.9655
Epoch 23/30
 - 16s - loss: 0.0874 - binary_accuracy: 0.9630
Epoch 24/30
 - 16s - loss: 0.0866 - binary_accuracy: 0.9645
Epoch 25/30
 - 17s - loss: 0.0837 - binary_accuracy: 0.9649
Epoch 26/30
 - 18s - loss: 0.0803 - binary_accuracy: 0.9677
Epoch 27/30
 - 18s - loss: 0.0824 - binary_accuracy: 0.9646
Epoch 28/30
 - 17s - loss: 0.0776 - binary_accuracy: 0.9677
Epoch 29/30
 - 19s - loss: 0.0744 - binary_accuracy: 0.9677
Epoch 30/30
 - 19s - loss: 0.0745 - binary_accuracy: 0.9683
```

```
# Codi 14:

model.save("twt1_1.h5")
hist=history.history
print(hist.keys())
acc=hist['binary_accuracy']
loss=hist['loss']
```

```
import matplotlib.pyplot as plt
epochs=range(1,len(acc)+1)

# Gràfic accuracy
plt.plot(epochs,acc,'#d9223a',label='Training_acc')
plt.title('Accuracy')
plt.legend()
plt.figure()

# Gràfic loss
plt.plot(epochs,loss,'#d9223a',label='Training_loss')
plt.title('Loss')
plt.legend()
plt.figure()
```

```
dict_keys(['loss', 'binary_accuracy'])
```

```
# Codi 15:

from sklearn.metrics import confusion_matrix
import itertools

predicted_labels = model.predict(np.stack(X_test))
cm = confusion_matrix(np.argmax(Y_test, axis=1),
                      np.argmax(predicted_labels, axis=1))
print('Matriu de confusió:\n',cm)

cm = cm.astype('float') / cm.sum(axis = 1)[:, np.newaxis]
plt.imshow(cm, cmap=plt.cm.Blues)
plt.title('Matriu de confusió normalitzada')
plt.colorbar()
plt.xlabel('Nivells reals')
plt.ylabel('Nivells predits')
plt.xticks([0, 1]); plt.yticks([0, 1])
plt.grid('off')
for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
    plt.text(j, i, format(cm[i, j], '.2f'),
             horizontalalignment='center',
             color='white' if cm[i, j] > 0.5 else 'black')
```

```
Matriu de confusió:
 [[2502  319]
 [ 312  408]]
```

Modificació 2

```
# Codi 16:

embed_dim = 128
lstm_out = 196
model = Sequential()
model.add(Embedding(input_dim=max_fatures,
```

```
                    output_dim=embed_dim, input_length = X.shape[1]))
model.add(SpatialDropout1D(0.4))
model.add(LSTM(units=lstm_out, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(2,activation='sigmoid'))
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['binary_accuracy'])

# Codi 17:

history=model.fit(X_train, Y_train,
                  epochs = 20,
                  validation_split=0.25,
                  batch_size=32,
                  verbose = 2)
```

```
WARNING:tensorflow:From /anaconda3/lib/python3.7/site-packages/tensorf
low/python/ops/math_ops.py:3066: to_int32 (from tensorflow.python.ops.
math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Train on 5391 samples, validate on 1797 samples
Epoch 1/20
 - 15s - loss: 0.4791 - binary_accuracy: 0.8017 - val_loss: 0.3959 - v
al_binary_accuracy: 0.8442
Epoch 2/20
 - 12s - loss: 0.3328 - binary_accuracy: 0.8599 - val_loss: 0.3647 - v
al_binary_accuracy: 0.8497
Epoch 3/20
 - 14s - loss: 0.2782 - binary_accuracy: 0.8893 - val_loss: 0.3668 - v
al_binary_accuracy: 0.8517
Epoch 4/20
 - 15s - loss: 0.2486 - binary_accuracy: 0.8992 - val_loss: 0.4129 - v
al_binary_accuracy: 0.8461
Epoch 5/20
 - 19s - loss: 0.2216 - binary_accuracy: 0.9127 - val_loss: 0.4084 - v
al_binary_accuracy: 0.8347
Epoch 6/20
 - 18s - loss: 0.1992 - binary_accuracy: 0.9224 - val_loss: 0.4316 - v
al_binary_accuracy: 0.8395
Epoch 7/20
 - 18s - loss: 0.1849 - binary_accuracy: 0.9282 - val_loss: 0.4377 - v
al_binary_accuracy: 0.8456
Epoch 8/20
 - 19s - loss: 0.1667 - binary_accuracy: 0.9349 - val_loss: 0.4685 - v
al_binary_accuracy: 0.8303
Epoch 9/20
 - 18s - loss: 0.1526 - binary_accuracy: 0.9412 - val_loss: 0.5474 - v
al_binary_accuracy: 0.8434
Epoch 10/20
 - 16s - loss: 0.1416 - binary_accuracy: 0.9399 - val_loss: 0.5884 - v
al_binary_accuracy: 0.8392
Epoch 11/20
 - 15s - loss: 0.1285 - binary_accuracy: 0.9482 - val_loss: 0.5737 - v
```

```
al_binary_accuracy: 0.8325
Epoch 12/20
 - 14s - loss: 0.1187 - binary_accuracy: 0.9539 - val_loss: 0.6257 - v
al_binary_accuracy: 0.8453
Epoch 13/20
 - 15s - loss: 0.1113 - binary_accuracy: 0.9535 - val_loss: 0.6668 - v
al_binary_accuracy: 0.8308
Epoch 14/20
 - 14s - loss: 0.1063 - binary_accuracy: 0.9571 - val_loss: 0.6712 - v
al_binary_accuracy: 0.8322
Epoch 15/20
 - 13s - loss: 0.0982 - binary_accuracy: 0.9593 - val_loss: 0.7489 - v
al_binary_accuracy: 0.8228
Epoch 16/20
 - 14s - loss: 0.0956 - binary_accuracy: 0.9613 - val_loss: 0.6816 - v
al_binary_accuracy: 0.8306
Epoch 17/20
 - 14s - loss: 0.0941 - binary_accuracy: 0.9609 - val_loss: 0.7584 - v
al_binary_accuracy: 0.8275
Epoch 18/20
 - 12s - loss: 0.0882 - binary_accuracy: 0.9637 - val_loss: 0.8453 - v
al_binary_accuracy: 0.8283
Epoch 19/20
 - 13s - loss: 0.0836 - binary_accuracy: 0.9657 - val_loss: 0.7862 - v
al_binary_accuracy: 0.8230
Epoch 20/20
 - 13s - loss: 0.0839 - binary_accuracy: 0.9635 - val_loss: 0.9568 - v
al_binary_accuracy: 0.8289
```

```
# Codi 18:

# Guardar model
model.save("twt1_2.h5")

# Representació
hist2=history.history
val_acc=hist2['val_binary_accuracy']
val_loss=hist2['val_loss']
acc=hist2['binary_accuracy']
loss=hist2['loss']
import matplotlib.pyplot as plt
epochs=range(1,len(acc)+1)

## Gràfic accuracy
import matplotlib.pyplot as plt
plt.plot(epochs,acc,'#d9223a',label='Training_acc')
plt.plot(epochs,val_acc,'#595959',label='Validation_acc')
plt.title('Training_vs Validation Accuracy')
plt.legend()
plt.figure()

## Gràfic loss
plt.plot(epochs,loss,'#d9223a',label='Training_loss')
plt.plot(epochs,val_loss,'#595959',label='Validation_loss')
```

```
plt.title('Training_vs Validation Loss')
plt.legend()
plt.figure()
```

```
<Figure size 432x288 with 0 Axes>
```

```
# Codi 19:

from sklearn.metrics import confusion_matrix
import itertools

predicted_labels = model.predict(np.stack(X_test))
cm = confusion_matrix(np.argmax(Y_test, axis=1),
                      np.argmax(predicted_labels, axis=1))
print('Matriu de confusió:\n',cm)

cm = cm.astype('float') / cm.sum(axis = 1)[:, np.newaxis]
plt.imshow(cm, cmap=plt.cm.Blues)
plt.title('Matriu de confusió normalitzada')
plt.colorbar()
plt.xlabel('Nivells reals')
plt.ylabel('Nivells predits')
plt.xticks([0, 1]); plt.yticks([0, 1])
plt.grid('off')
for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
    plt.text(j, i, format(cm[i, j], '.2f'),
             horizontalalignment='center',
             color='white' if cm[i, j] > 0.5 else 'black')
```

```
Matriu de confusió:
 [[2522  299]
 [ 316  404]]
```