INTEL·LIGÈNCIA ARTIFICIAL

Creació d'una intel·ligència artificial
per traduir de llenguatge de signes a text

Abstract

It is undeniable that the emergence of artificial intelligence and technological advances in this field have a significant impact on our daily days, indeed, it is said that artificial intelligence is the most important technological revolution since computer science was discovered.
This research exposes the process of developing an application with artificial intelligence from scratch. The main goal of this project is to create a neural network model able to translate to text the predominant sign language in the United States and Canada, also known as the ASL alphabet (Alphabet Sign Language).
The research starts by presenting what artificial intelligence is and explaining all the theoretical concepts necessary to develop the artificial intelligence model, focusing mainly on deep learning and artificial intelligence applied in image classification. Afterward, there is an explication of each of the technologies used during the programming process, the Python language and the library named TensorFlow being the most important. Consecutively, all steps followed during the coding process and the implementation of the artificial intelligence model, are explained.
Successively, the project shows the final result of the application, which is used to draft the conclusions of the research. The document ends with the acknowledgments, the webography, and the appendices.

Resum

És innegable que l'aparició de la intel·ligència artificial i els avenços tecnològics en aquest camp tenen un impacte important en el nostre dia a dia, de fet, es diu que la intel·ligència artificial és la revolució tecnològica més important des que es va descobrir la informàtica. Aquest Treball de Recerca exposa el procés de desenvolupament d'una aplicació amb intel·ligència artificial des de zero. L'objectiu principal d'aquest projecte és crear una xarxa neuronal capaç de traduir a text la llengua de signes predominant als Estats Units i Canadà, també conegut com a abecedari ASL (Alphabet Sign Language).
El treball comença exposant què és la intel·ligència artificial i explicant tots els conceptes teòrics necessaris per desenvolupar el model d'intel·ligència artificial, centrant-se principalment en l'aprenentatge profund i la intel·ligència artificial aplicada a la classificació d'imatges. A continuació, es fa una explicació de cadascuna de les tecnologies utilitzades durant el procés de programació, sent el llenguatge Python i la biblioteca anomenada TensorFlow els més importants. Consecutivament, s'expliquen tots els passos seguits durant el procés de codificació i implementació de la intel·ligència artificial.
Seguidament, el projecte mostra el resultat final de l'aplicació, que serveix per redactar les conclusions del treball. El document acaba amb els agraïments, els recursos electrònics i els annexos.

Índex

Capítol 1

Presentació

1.1 Motius per fer el treball

El primer contacte que vaig tenir amb el concepte d'intel·ligència artificial va ser durant un curs de la Fundació Catalana La Pedrera anomenat Bojos per la Supercomputació on durant dues sessions, se'ns van presentar de manera sintetitzada la intel·ligència artificial. Amb aquesta introducció, em vaig poder adonar de quina manera aquest nou sistema computacional pot suposar una revolució en la manera de programar, i per tant, com pot canviar el nostre dia a dia.
No obstant això, malgrat la importància que la intel·ligència artificial està agafant en la nostra societat, la desconeixença sobre aquest terme és més extensa del que creia. És per aquesta raó que vaig decidir enfocar el meu treball cap a l'àmbit de la intel·ligència artificial. La meva intenció amb aquest treball és, per una banda comprendre què és aquest nou sistema de software, com treballa i per què es diu que pot suposar una revolució, mentre que per altra banda, aquest treball també té com a objectiu poder explicar a la gent del meu voltant el significat d'aquest nou terme que cada vegada és més comú i com pot influir en la vida quotidiana de les persones.

1.2 Organització i objectius del treball

Quan vaig decidir enfocar el treball cap aquest àmbit, el coneixement que tenia d'aquest camp era molt baix. El fet de partir des de tan poc coneixement, va fer que no pogués determinar uns objectius des de l'inici sinó que a mesura que avançava el treball, vaig anar construint i reenfocant aquests objectius.
El primer objectiu es va basar a comprendre què és la intel·ligència artificial i aprendre aquells conceptes necessaris per poder crear-ne una. Un cop acabat aquest procés, amb una durada aproximada d'un mes, els objectius del treball es van reenfocar.
Un cop compresos els coneixements i després d'haver seguit explicacions de com fer intel·ligències artificials, vaig proposar-me crear la meva pròpia intel·ligència artificial partint únicament dels coneixements adquirits durant el mes anterior. Afiliat a aquest objectiu, la meva intenció era crear una intel·ligència artificial beneficiosa per les persones i que

d'alguna manera, en un futur pogués ajudar a un sector de la població. Aquest procés va tenir una durada aproximada de tres mesos.
Un cop acabat aquest procés, es va fixar un nou objectiu, crear una aplicació per poder implementar la intel·ligència artificial, tot i que soc conscient que no disposava ni dels recursos computacionals ni dels coneixements perquè fos realment potent i ben desenvolupada. La intenció d'aquesta aplicació és mostrar, per una banda, com funciona la intel·ligència artificial desenvolupada i per l'altra, ensenyar la manera d'implementar una intel·ligència artificial.

Capítol 2

Introducció

El terme d'intel·ligència artificial neix oficialment l'any 1956 en mans de l'informàtic John McCarthy, durant la conferència de Darmouth. En aquesta conferència[1] John McCarty va convidar molts dels grans investigadors de l'època per parlar sobre el nou terme- la intel·ligència artificial- i al mateix temps definir les directrius i les actuacions futures en aquest àmbit. El 1957 es va crear la primera xarxa neuronal artificial que imitava la ment humana, a mà de Frank Rosenblat, i no va ser fins al 1966 que Joseph Weizenbaum va crear el primer *chatbot*, el primer programa capaç de processar el llenguatge natural humà.

La intel·ligència artificial és una important revolució que suposa un forma completament nova de software. Es diu que pot suposar l'avenç més rellevant en la tecnologia en els últims segles. De fet, Sundar Pichai[2] afirma que l'impacte de la intel·ligència artificial en la història de la humanitat és comparable amb el de l'electricitat i el foc. En pocs anys, s'ha implementat en diversos sectors com el de la sanitat, indústria i finances, ocupant també serveis d'atenció al client i arribant fins i tot a algunes cases en forma d'assistent virtual, més conegut com l'Alexa.

Malgrat tot, encara ara, se'ns fa difícil definir el concepte d'intel·ligència artificial, ja que no hi ha una definició acceptada per tots els experts. En la forma més simple, es pot descriure la intel·ligència artificial (IA) com a un sistema de computació que busca simular processos pròpiament del cervell humà en sistemes informàtics. En altres paraules, la intel·ligència artificial té com a objectiu aconseguir que una màquina percebi el seu entorn, aprengui i raoni buscant els millors algorismes per tal de solucionar els problemes plantejats.

Generalment, trobem dos tipus d'intel·ligència artificial, les intel·ligències artificials dèbils, que són totes aquelles que només poden complir amb un conjunt molt limitat de tasques, i les intel·ligències artificials fortes, que són totes aquelles capaces d'aplicar-se en un conjunt molt ampli de tasques, amb dominis molt diferents.[3]

[1] La trobada va tenir una durada aproximada de dos mesos i va ser la reunió sobre intel·ligència artificial més important que s'havia produït fins al moment.

[2] Director executiu de Google.

[3] Avui en dia, totes les intel·ligències artificials són dèbils.

2.1 Aprenentatge automàtic

La intel·ligència artificial utilitza diferents tècniques entre les quals, destaca l'aprenentatge automàtic.[4]

L'aprenentatge automàtic és el procés que permet que l'ordinador pugui aprendre a partir de dades, per tal d'identificar patrons i desenvolupar algoritmes per cada problema i així fer una predicció per cada cas particular.

Figura 2.1: Programa clàssic

A la imatge 2.1, es pot observar com funciona un programa clàssic. El programa rep dues entrades, unes dades d'entrada i les normes, dictades en les instruccions del programa, a partir de les quals aconsegueix extreure una resposta.

Figura 2.2: Programa aprenentatge automàtic

La imatge 2.2, ens mostra com funciona un programa amb aprenentatge automàtic. En aquest cas, les dues entrades del nostre programa són les dades d'entrada,[5] i les respectives respostes,[6] fent així que el què hagi d'extreure de la informació que no sigui la resposta sinó les normes. Això ens permet que més endavant, tan sols donant al programa les dades d'entrada, aquest apliqui les normes extretes prèviament i torni una resposta.

[4]Més conegut com a *machine learning*. (ML)
[5]També anomenades característiques, paràmetres, atributs o *features*.
[6]També s'anomenen etiquetes o *labels*.

```
def celsius_a_fahrenheit(c):
    return (c * 1.8) + 32

c=50
f = celsius_a_fahrenheit(c)
print("50ºC són " +str(f)+ "ºF")

50ºC són 122.0ºF
```

Figura 2.3: Exemple d'un programa clàssic

La imatge 2.3 mostra un programa programat de manera clàssica, capaç de passar de graus Celsius a graus Fahrenheit.
En color taronja es poden veure les dades d'entrada del programa, en aquest cas, el valor de graus Celsius que volem convertir. La part verda senyalitza les normes del programa, l'equació que ens permet passar d'una unitat a l'altra. Finalment, es pot observar la resposta del problema en color blau.

```
celsius = np.array([0, 3, 4, 5, 8, 9, 10, 12, 15, 18, 20, 22, 25, 28, 30, 33, 35, 38, 40], dtype=float)
fahrenheit = np.array([32, 37, 39, 41, 46, 48, 50, 54, 59, 64, 68, 72, 77, 82, 86, 91, 95, 100, 104], dtype=float)

model= tf.keras.Sequential([
  tf.keras.layers.Dense(3, input_shape=[1]),
  tf.keras.layers.Dense(3),
  tf.keras.layers.Dense(1)
])

model.compile(optimizer=tf.keras.optimizers.Adam(0.01),
              loss='mean_squared_error',
 )

history=model.fit(
    celsius, fahrenheit,
    epochs=350)

resultat=model.predict([50])
print("50ºC són"+str(resultat)+"ºF")

50ºC són[[121.870735]]ºF
```

Figura 2.4: Exemple d'un programa amb aprenentatge automàtic

A la imatge 2.4 es pot observar un programa amb aprenentatge automàtic que realitza la mateixa funció que el programa anterior. En color taronja, s'hi poden veure les dades d'entrada del programa, en aquest cas diferents valors de graus Celsius. La part verda senyalitza els valors en graus Fahrenheit de les dades d'entrada. A partir d'aquestes dades, el model extreu unes normes, que són aplicades per calcular la resposta.

En el cas del programa clàssic, la probabilitat que cometi un error és pràcticament nul·la, en canvi, en el cas de l'aprenentatge automàtic, sempre hi haurà un marge d'error, per

això es busca aconseguir l'exactitud més gran possible.[7]

2.2 Aprenentatge profund

L'aprenentatge profund[8] com es pot apreciar en la imatge 2.5, és un cas particular de l'aprenentatge automàtic, probablement el més exitós, ja que és capaç de processar grans volums de dades en molt poc temps.

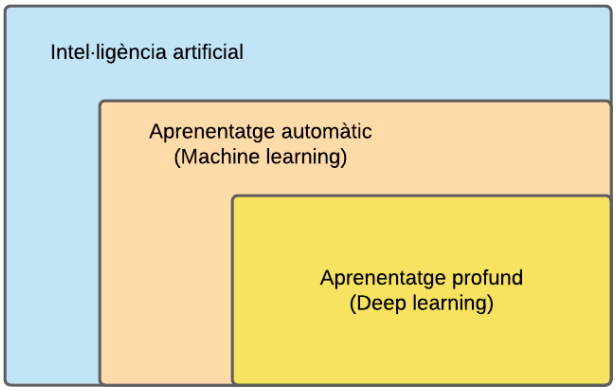


Figura 2.5: Esquema sobre la intel·ligència artificial

Tant l'aprenentatge profund com l'aprenentatge automàtic utilitzen xarxes neuronals que imiten la forma d'aprendre del cervell humà. La seva diferència es troba en el tipus d'algorismes que empren.
L'aprenentatge automàtic compta amb una xarxa neuronal tan limitada que es diu que fa ús d'arbres de decisió. Per altra banda, la forma d'aprendre en el cas de l'aprenentatge profund és més semblant a la forma d'aprendre de les persones, ja que usa xarxes neuronals més evolucionades.
En els esquemes 2.6 i 2.7, es pot apreciar com l'única diferència entre l'aprenentatge automàtic i l'aprenentatge profund, es troba en la forma com processen la informació.

[7]Més conegut com a *accuracy*. Es calcula dividint les prediccions correctes entre el nombre total d'exemples utilitzats.
[8]Més conegut com a *deep learning*.

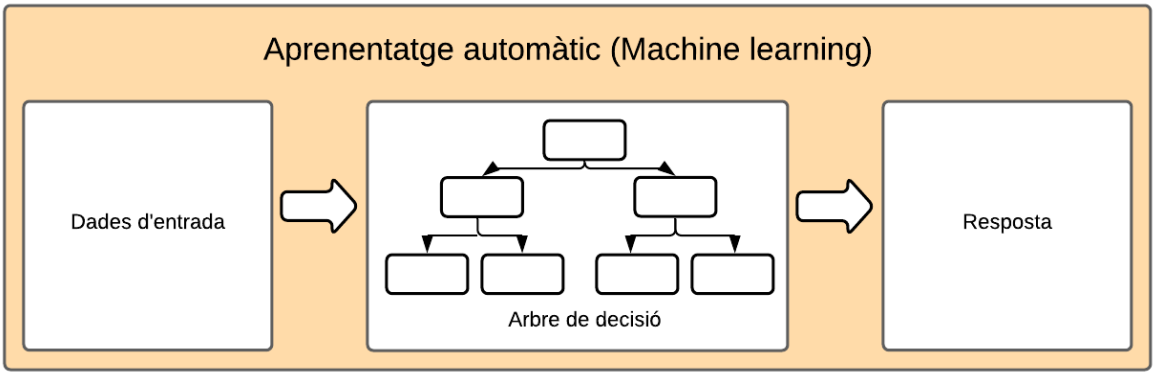


Figura 2.6: Esquema sobre l'aprenentatge automàtic

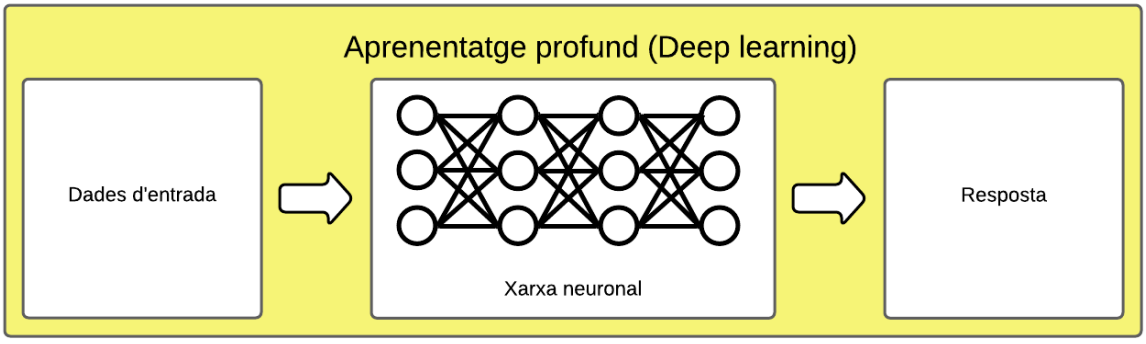


Figura 2.7: Esquema sobre l'aprenentatge profund

Capítol 3

Xarxes neuronals

Com s'ha dit anteriorment, tant en l'aprenentatge automàtic com en l'aprenentatge profund s'utilitzen les xarxes neuronals. Una xarxa neuronal és un conjunt de nodes (neurones) connectades entre si, formant una estructura de capes d'aprenentatge que simula un cervell humà. Aquestes capes són específicament dissenyades per reconèixer patrons. Com que aquests patrons són vectors numèrics, qualsevol altre tipus d'informació, ja siguin imatges, sons, paraules... ha de ser traduït.
La principal virtut de les xarxes neuronals és que, gràcies a un entrenament previ, poden aprendre a executar tasques sense la necessitat d'estar específicament programades.
En les xarxes neuronals distingim tres tipus de capes diferents, la capa d'entrada o *input layer*, les capes amagades o *hidden layers* i la capa de sortida o *output layer*.

En la imatge 3.1, els cercles representen neurones mentre que cada fila de cercles d'un mateix color, són les capes.[1]

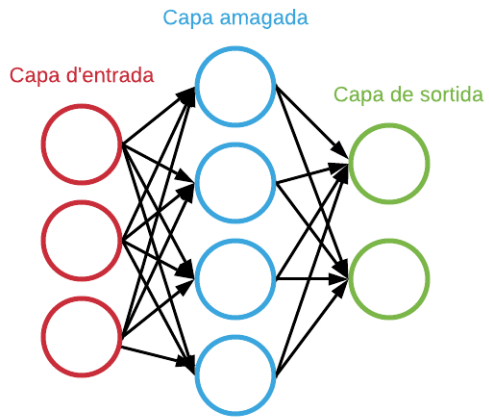


Figura 3.1: Xarxa neuronal

Capa d'entrada: La capa d'entrada, és la primera capa de la xarxa neuronal, així que ha de tenir el mateix nombre de neurones que de dades d'entrada. És a dir, si nosaltres volem que les dades d'entrada siguin imatges de 4096 píxels, aquesta capa haurà de tenir 4096 neurones.

Capes amagades: Normalment ens trobem amb més d'una capa amagada. Les capes amagades són un conjunt de capes intermèdies entre

[1]En aquest cas, com que totes les neurones d'una capa estan connectades amb totes les neurones de la capa següent, diem que tenim una capa totalment connectada. En el cas contrari, diem que tenim una xarxa neuronal parcialment connectada. Això passa quan es perd alguna connexió entre neurones.

la capa d'entrada i la capa de sortida. Poden contenir totes les neurones que facin falta. Diem que com més capes amagades i més neurones tingui una xarxa neuronal, més complexa és la intel·ligència artificial, i per tant, tindrà més capacitat de processament de dades.

Capa de sortida: La capa de sortida és l'última capa, la qual recull el resultat del processament de la xarxa neuronal.

3.1 Neurones

La neurona és la unitat bàsica que forma una xarxa neuronal. La imatge 3.2 mostra una representació d'una neurona. Per entendre com funciona, es divideix en tres parts:

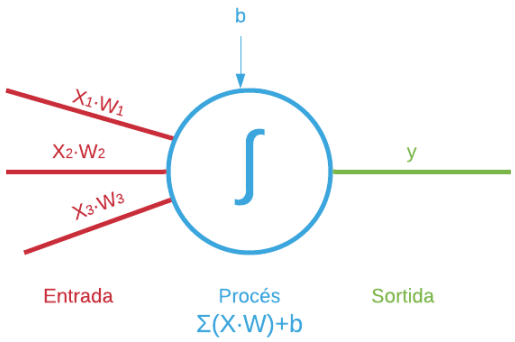


Figura 3.2: Representació d'una neurona

1. La part vermella és la part d'entrada on arriben les connexions amb la capa anterior. En aquest cas en són tres. Aquestes connexions són les que passen informació a la neurona, en forma de número.

2. La part blava és la part principal de la neurona. És on es processa la informació, a partir de normes i funcions.

3. La part verda és la part de sortida. Recull tota la informació de la neuronal per passar-la a la següent neurona.

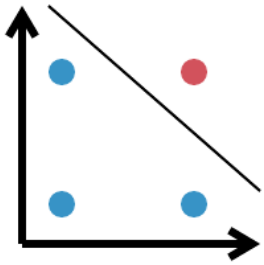
Cada connexió de la neurona, porta un valor X, que és el valor o la informació que ens està enviant la neurona passada, i un valor W. Aquest valor s'anomena el pes, que com indica el nom, diu la importància que té la connexió. És un valor que ajuda a donar-li més importància a una connexió que a una altra.[2]
Un cop la neurona ha rebut el valor anomenat X i els pesos, internament realitza la funció $\sum(X \cdot W)+b$. És a dir, fa una suma ponderada per cada connexió i suma totes les connexions (X1· W1 + X2· W2 + X3· W3), al resultat li suma el valor b. Aquest valor s'anomena biaix i és un nombre que fomenta que algunes neurones s'activin amb major facilitat que altres. Es representa com a una altra connexió de la neurona, però que permet assignar-li

[2]Aquests pesos són els paràmetres del model. Són els valors que podem modificar o ajustar per tal que la nostra xarxa neuronal pugui aprendre i ens torni el resultat esperat.

el valor que nosaltres determinem.
Si ens fixem en la funció que fa una neurona, es pot observar com correspon a la funció lineal d'una recta: y = (X·W)+b. Aquesta funció consta d'unes variables d'entrada que defineixen la recta -en aquest cas els valors X, la informació de la neurona anterior, i els valors W, els pesos-, un terme independent, que serveix per moure la recta verticalment, -en el cas d'una neurona aquest valor és el biaix-, i el valor y, el valor que es busca.

Però, com pot ser útil una neurona per codificar informació?

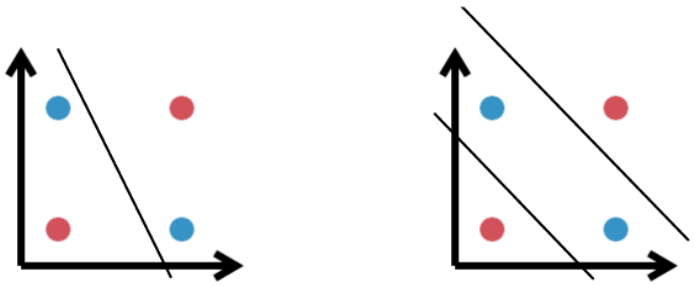
A les gràfiques següents hi ha quatre punts que representen les dades d'entrada. Cada punt té associat un color, o blau o vermell i un valor depenent del color. Els punts de color blau tenen associats el valor d'1, mentre que el valor dels punts vermells és 0. La intel·ligència artificial ha d'intentar buscar una funció que s'adapti a les dades d'inici per tal de traçar una línia que separi els punts vermells amb els punts blaus i així, quan se li entri un valor que no ha vist anteriorment, pugui declarar a través d'aquest criteri, si el nou valor val 0 o 1.



La figura 3.3 correspon a una porta OR amb tres punts blaus, de valor 1 i un punt vermell de valor 0. En aquest cas la intel·ligència artificial crearia la funció lineal que separa el punt vermell dels blaus, fent que si s'entra un nou valor, pot saber si val 0 o 1 només situant-lo a la gràfica.

Figura 3.3: Representació porta OR

No obstant això, no tots els problemes es poden resoldre amb una única neurona, com passa en el cas d'una porta XOR. Tal com es pot observar en la imatge (a) de la figura 3.4, resoldre una porta XOR amb una única neurona, és a dir, amb una única funció lineal és impossible.
Per resoldre aquest problema, són necessàries dues funcions lineals, tal com es mostra a la imatge (b) de la figura 3.4. En aquests casos més complexes és quan necessitem una xarxa neuronal.

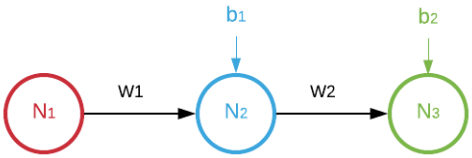


(a) Representació d'una porta XOR (b) Solució d'una porta XOR

Figura 3.4: Porta XOR

3.2 Funcions d'activació

En una xarxa neuronal es connecten diferents neurones entre elles. Si cada neurona realitza una funció lineal, el que fa una xarxa neuronal doncs, és ajuntar diferents funcions lineals. El problema d'això, és que matemàticament, si s'ajunten diferents funcions lineals, com a sortida s'obté una funció lineal. Per veure-ho més clar, ens podem fixar en l'exemple següent:



La imatge 3.5 mostra una xarxa neuronal formada per tres capes amb una neurona per capa. Fixem-nos en la funció que realitza cada neurona:

Figura 3.5: Xarxa neuronal

En aquest cas, com que la N1 és la neurona d'entrada, no realitza cap operació, simplement passa la informació -que anomenarem X- a la N2.
La N2 realitza l'equació vista anteriorment: N2=W1·X+b1, i passa el nou valor calculat a la següent neurona.
Es pot dir doncs, que la funció que realitza la N3: N3=W2·N2+b2.
Si se substitueix el valor N2 a l'equació: N3=W2(W1·X+b1)+b2
Per tant queda que: N3=W2(W1·X)+W2·b1+b2
Si s'arregla: X(W2·W1)+W2·b1+b2
Per veure-ho més clar, podem dir: W'=W2·W1 i b'=W2·b1+b2 Si se substitueixen les noves variables a la fórmula queda que: N3=W'·X+b', el que torna a ser una funció lineal.

Així doncs, malgrat crear una xarxa neuronal, tots els problemes que no es puguin resoldre

amb funcions lineals, la intel·ligència artificial no els podrà resoldre.
Per evitar que això passi, s'utilitza les funcions d'activació.

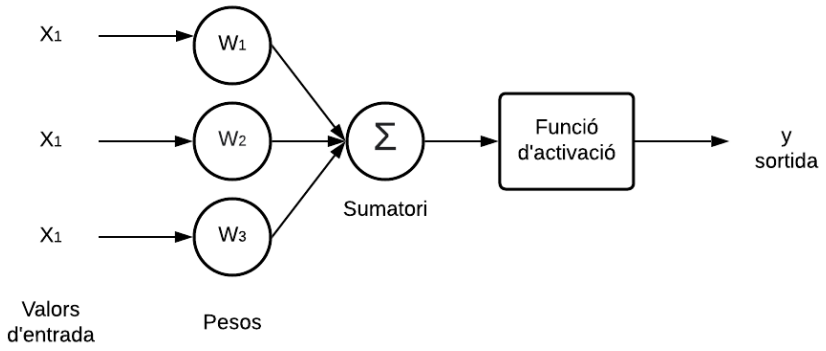
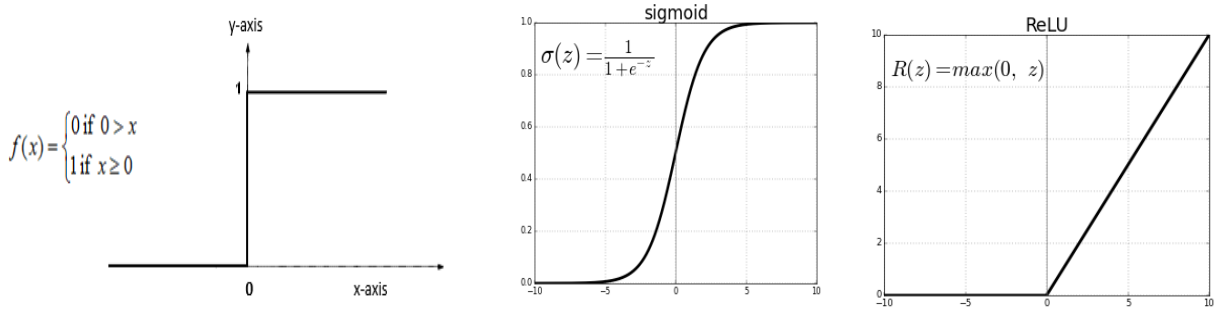


Figura 3.6: Funcions d'activació

Les funcions d'activació distorsionen les funcions lineals de cada neurona convertint-les en no lineals. Aquestes funcions s'apliquen al valor final de cada neurona. Per entendre-ho millor, vegem diferents exemples de funcions d'activació.
A la figura 3.7, es mostra les tres funcions d'activació més utilitzades.



(a) Funció escalonada (b) Funció sigmoide (c) Funció ReLu

Figura 3.7: Funcions d'activació

La imatge (a) de la figura 3.7, mostra funció **escalonada**. El que fa aquesta funció és un canvi de variable. Mentre el valor sigui més petit de 0, el valor de sortida de la neurona val 0, si el valor val 0 o és més gran a 0, val 1. Rep aquest nom, ja que el canvi de variable no es produeix gradualment, i per tant, la funció té la forma d'escaló.

Una altra funció que coneixem és la funció **sigmoide**, la imatge (b) de la figura 3.7. En aquest cas la distorsió fa que els valors més grans es limitin a un valor, en el cas de la imatge a 1, i que els valors més petits els limitin a un altre valor, en el cas de la imatge a 0. En aquest cas, la funció sigmoide té la forma similar a "S".

La funció **ReLU**, que pertany a la funció de la imatge (c) de la figura 3.7, bàsicament es

comporta com una funció lineal per aquells valors positius, i constant de 0 per aquells valors d'entrada negatius.

Capítol 4

Passos per crear una intel·ligència artificial

La creació d'una intel·ligència artificial utilitzant l'aprenentatge automàtic consta d'uns passos genèrics.

1. Obtenir i preparar les dades d'entrada
2. Crear la xarxa neuronal
3. Entrenar el model
4. Avaluar el model

Per explicar cada pas, s'enfocarà en un problema de classificació d'imatges.

4.1 Obtenir i preparar les dades d'entrada

El primer pas consisteix a obtenir i preparar les dades amb les quals el model aprendrà. Aquest és el pas que més feina comporta i el que requereix més temps.
La qualitat i la mida de les dades d'entrada importen més que els algorismes utilitzats, per això s'ha de procurar fer ús d'un bon conjunt de dades d'entrada, ja que el fet de no comptar amb les dades adequades pot fer que el nostre model tingui un rendiment menor. Si les dades que utilitzem no són les correctes, el model acabarà creant funcions que no s'ajusten a res, fent que malgrat tenir una bona xarxa neuronal, no hagi après correctament i per tant no pugui desenvolupar-se plenament.
Es diu que com més complexa és la tasca, més dades es necessiten. No obstant això, la qualitat de les dades és important. No serveix per res tenir moltes dades si no són vàlides. Es pot dir que unes dades són bones si compleixen la seva tasca prevista. Si, per exemple, es vol aconseguir un model que diferenciï entre gossos i gats, s'ha de procurar que les dades continguin imatges de gossos i imatges de gats, de mides, races i colors diferents.

A l'hora de crear les dades d'entrada, sobretot quan es tracten d'imatges i vídeos, s'acostuma a seguir uns passos:

1. Recol·lecció
2. Filtrat
3. Neteja
4. Anotació

Recol·lecció: La recol·lecció s'enfoca en recol·lectar les dades, en aquest cas imatges, ja sigui buscant-les o creant-les un mateix. Això depèn de cada projecte i els recursos disponibles, ja que aquestes dades han d'abastar totes les possibilitats.

Filtrat: És molt probable que un cop recol·lectades totes les imatges, se'n descartin algunes. Generalment les imatges que tinguin poca resolució, mala il·luminació o bé que siguin borroses. No obstant això, depenent del projecte aquestes imatges poden ajudar a crear models més complexes.

Neteja: Modificar o editar les imatges. És útil sobretot quan la quantitat de dades no és suficientment alta. En aquests casos, és millor editar algunes imatges que descartar-les. Les edicions estan orientades a afegir defectes a les imatges per aconseguir models més eficients.

Anotació: Aquest pas depèn del tipus de problema a resoldre. L'anotació consisteix a etiquetar les imatges.[1] Depenent del tipus de problema a resoldre, aquesta etapa pot ser tan simple com definir una o més etiquetes per cada imatge (en el cas d'un classificador d'imatges) o pot consistir a marcar cada objecte de la imatge (en el cas que el problema sigui a identificar tots els objectes d'una imatge).

Un cop es tenen totes les dades, s'han de dividir. Una part s'utilitza per a l'entrenament, el que s'anomena dades d'entrenament i una altra part per l'avaluació, és a dir, per veure com de bé treballa el model. La proporció d'aquesta divisió, sol ser 80/20, és a dir, si un conjunt consta de 100 imatges, 80 s'utilitzaran per a l'entrenament i 20 per l'avaluació. El que s'aconseguix amb aquesta divisió és poder controlar com treballa el model amb dades que no ha vist anteriorment.

[1] Les etiquetes és el que el model ha de predir, és a dir, són les solucions dels problemes plantejat.

4.2 Crear el model

Un cop es tenen les dades d'entrada, s'ha de crear la xarxa neuronal de la intel·ligència artificial. Un dels principals problemes de les xarxes neuronals és la definició de la seva arquitectura. Totes les xarxes neuronals tenen una capa d'entrada i una de sortida, que rep els estímuls externs, el problema són les capes amagades.
Les xarxes neuronals amb menys neurones i capes amagades, són preferibles a les xarxes neuronals complexes, ja que l'entrenament és més ràpid i tenen una major capacitat de generalitzar, és a dir, buscar algorismes que es puguin utilitzar en la majoria dels casos. Ara bé, no tots els problemes es poden resoldre amb xarxes neuronals simples.
El nombre de les capes amagades es troba a partir d'un procés de prova i error. És a dir, provant diferents estructures fins a trobar la xarxa neuronal que treballi millor.

4.2.1 Xarxes neuronals convolucionals

Les xarxes neuronals es poden classificar segons la tipologia o l'arquitectura. Unes xarxes que són especialment importants en el processament d'imatge i la categorització són les xarxes neuronals convolucionals.[2]
Les xarxes neuronals convolucionals són molt potents en l'anàlisi d'imatges. Es comporten de manera molt similar a com ho fan les neurones de l'escorça visual primària d'un cervell humà i compten amb la capacitat d'identificar i desxifrar els patrons de les imatges.
Per entendre millor com funcionen aquestes xarxes neuronals, fixem-nos en el procés de visió que seguim els humans:

Si se't mostra la imatge d'una cara, només mirant-la sabràs que pertany a una cara. Això ho saps, ja que escanejant la imatge, has detectat alguns elements que saps que conformen una cara, com poden ser els ulls, nas i boca.
I com saps que una part de la imatge correspon a un ull?
Perquè has detectat característiques que conformen un ull, com la línia de pestanyes, la pupil·la, l'iris...

I com has reconegut cada part de la imatge?
Perquè ets capaç de detectar patrons circulars, canvis de contrastos, textures, vores...

Així doncs, el procés per detectar una imatge comença identificant elements bàsics i generals, com les vores, textures, canvis de contrastos..., que més endavant es combinen formant patrons més complexos, en l'exemple anterior seria l'ull.

Per detectar els elements bàsics d'una imatge, les xarxes neuronals convolucionals compten amb una operació matemàtica anomenada convolució. Una convolució és una operació que jugant amb els valors dels píxels és capaç de crear noves imatges on es ressalten unes característiques concretes.

[2] *Convolutional Neutral Net* o CNN.

Figura 4.1: Exemple imatge amb convolució

A la imatge 4.1, es pot veure un exemple d'una imatge a la qual se li ha aplicat una convolució per tal que la silueta del gos sigui més evident.

Per entendre com funcionen les convolucions, posem el cas d'una imatge d'entrada en blanc i negre. El model interpreta aquesta imatge com una matriu de dues dimensions, que representen els píxels. Com que la imatge és en blanc i negre, cada píxel rep un valor de 0 a 255, sent 0 el color negre i 255 el color blanc.[3]

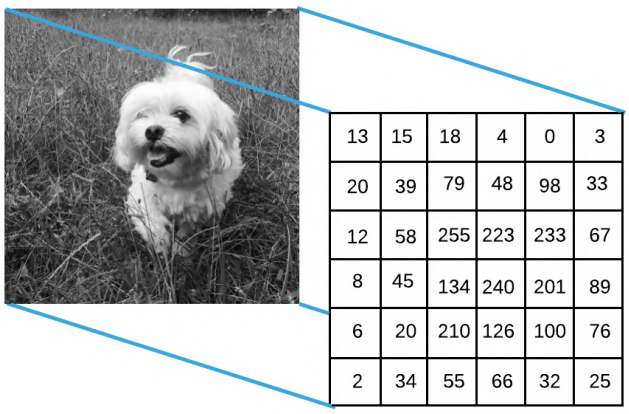


Figura 4.2: Exemple d'una imatge i la seva matriu

A la figura 4.2, es pot veure un exemple d'una imatge 6x6 en blanc i negre i el valor dels seus píxels.[4]

[3]Per treballar millor amb els valors dels píxels, es transformen els valors en valors entre 0 i 1, dividint-los per 255.

[4]Els valors de l'exemple són triats a l'atzar, és a dir, en aquest cas, no coincideixen amb els valors que realment tenen els píxels de la imatge.

La idea de la capa convolucional és aplicar filtres a la imatge per tal de detectar patrons importants dins de la imatge. Per fer-ho, es crea una altra taula de valors anomenada filtre o *Kernel*, tal com s'observa a la imatge 4.3.

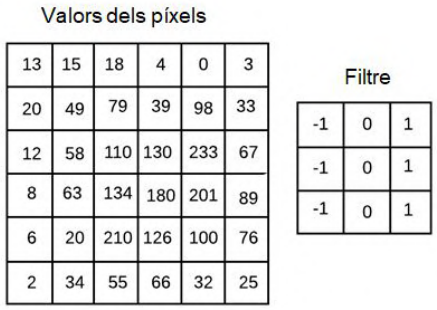


Figura 4.3: Exemple d'un filtre

Els nombres de dins del filtre és el que la intel·ligència va variant durant l'entrenament per tal de millorar la seva exactitud. Canviar els nombres de dins el filtre suposa obtenir com a sortida la mateixa imatge però amb unes característiques diferent en cada cas. És a dir, si per exemple, es posa un 1 al mig i la resta dels espais amb un 0, s'obté una rèplica de la imatge d'entrada. Si a cada quadradet del filtre se li atribueix el valor d'1/9, s'obté la mateixa imatge d'entrada però desenfocada. Si s'omple la primera columna amb el valor -1, la del mig amb 0 i la de la dreta amb 1, com es pot apreciar a la figura 4.3, s'aconsegueix la mateixa imatge d'entrada amb els canvis de contrastos marcats, el que serveix per distingir vores verticals.
En el cas que es vulgui aplicar el filtre de la imatge 4.3, al píxel de la taula que té el valor 110, el primer pas és situar el filtre centrat sobre el píxel (en aquest cas el de color verd). Un cop fet això, només s'utilitzen els valors de dins el filtre[5] delimitats pel quadrat taronja.

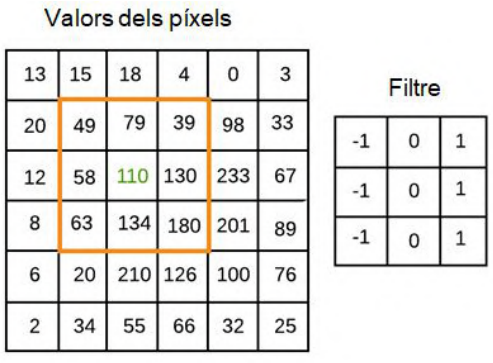


Figura 4.4: Primer pas d'una convolució

[5]En aquest cas, és 3x3, ja que és la mida determinada prèviament.

Per calcular el nou valor del píxel, amb un valor inicial de 110, es multiplica cada valor dels píxels amb els seus valors corresponents del filtre. Un cop es tenen els nou nous valors, se sumen. És a dir, en el cas de la figura 4.4, aquesta operació seria: 49·(-1) + 58·(-1) + 63·(-1) + 79·0 + 110·0 + 134·0 + 39·1 + 130·1 + 180·1, el que dona un resultat de 179. Per tant, el valor del nou píxel en comptes de ser 110 és 179. Si fem això amb tots els píxels, obtenim una nova imatge.

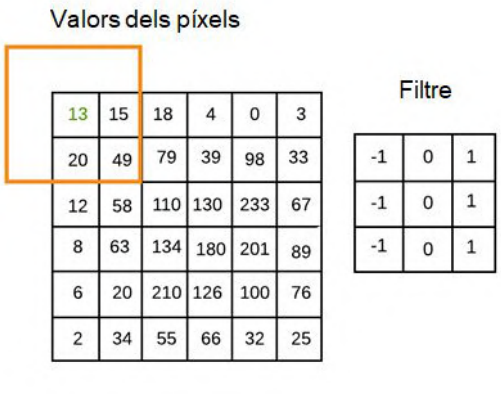


Figura 4.5: Problema comú en les convolucions

S'ha dit anteriorment que per trobar el nou valor d'un píxel, s'ha de col·locar el filtre de tal manera que aquest píxel quedi al mig. El problema sorgeix quan s'ha de calcular el valor d'un píxel situat als marges de la imatge, ja que com s'aprecia a la imatge 4.5, hi ha una part del filtre que no cobra cap valor.

En aquest cas, hi ha dues opcions. La primera opció és no utilitzar aquests píxels en la convolució. El problema d'aquesta opció és que es perd informació, ja que la nova imatge és més petita que la imatge d'entrada.

Una altra opció, més utilitzada, s'anomena *zero-padding*. En aquest cas, s'afegeixen píxels amb el valor de 0 pel voltant de la imatge, com es pot apreciar a la imatge 4.6.

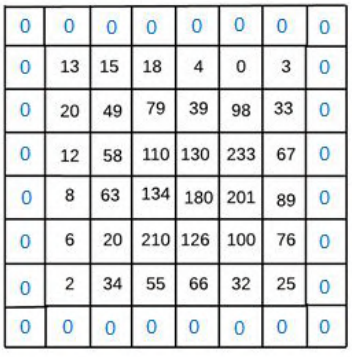


Figura 4.6: Exemple de zero-pooling

D'aquesta manera, s'aconsegueix omplir els espais del filtre que abans no tenien cap valor amb el valor 0, i per tant, pot calcular sense cap problema el nou valor del píxel, obtenint així una nova imatge sense perdre informació.
Les xarxes neuronals convolucionals es caracteritzen també per fer una altra operació matemàtica anomenada max pooling, que consisteix a reduir la mida de la imatge d'entrada. Per tal de dur a terme el max pooling, s'ha de triar dues coses, la mida d'una quadrícula o taula, el que s'anomena *pool* i un *stride*. L'*stride* és el nombre de píxels que la taula es desplaça, cap a la dreta o cap avall durant cada iteració. Com més gran és aquest valor, menor serà la imatge resultant.

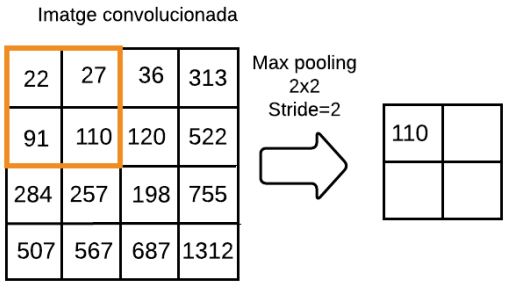


Figura 4.7: Exemple de max pooling (I)

A la imatge 4.7 es pot observar com s'aplica la funció max pooling en una imatge. El primer pas és situar la graella de la qual ja s'ha decidit les mides, en aquest cas de 2x2, sobre la imatge. Entre els nombres que queden dins la graella, es tria el valor més gran, en aquest cas 110, i se situa a la primera casella. Seguidament, es mou la graella, depenent de l'stride. En aquest cas com que tenim un stride 2, s'ha de moure dues posicions, tal com quedaria a la imatge 4.8.Com que els nombres són 36, 313, 120 i 522, es resumeix amb 522, ja que és el més gran.

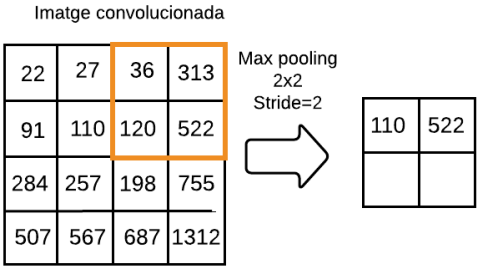


Figura 4.8: Exemple de max pooling (II)

Aquest procediment es repeteix fins a haver passat per a cada part de la imatge, obtenint un resultat com el de la figura 4.9.

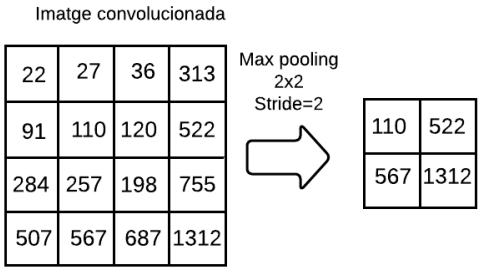


Figura 4.9: Exemple de max pooling (III)

4.2.2 Imatges en color

La diferència entre una imatge en color i una en blanc i negre és que en el cas de les imatges sense color els ordinadors l'interpreten com a una matriu de dues dimensions, en canvi, quan les imatges són amb color, les interpreta com a una matriu de tres dimensions.

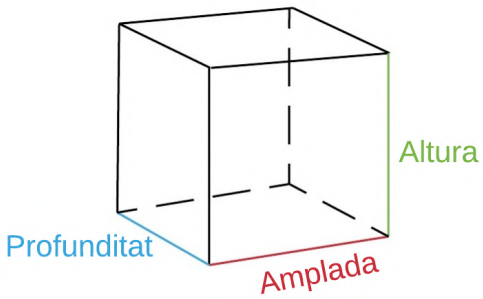


Figura 4.10: Representació d'una imatge en color

Una imatge en blanc i negre consta d'una amplada i una altura, així doncs, l'únic que els diferencia és la profunditat que ve marcada pel nombre de canals de colors. Moltes imatges poden ser representades per tres canals de diferents colors, el vermell, el verd i el blau, aquestes imatges s'anomenen RGB.[6] En la imatge 4.11, podem veure una representació d'una imatge RGB, que té tres capes, una vermella, una verda i una blava. Totes tres són de la mateixa mida.

[6] R=red, G=green, B=blue.

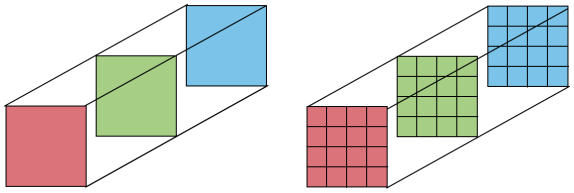


Figura 4.11: Representació d'una imatge RGB

4.2.3 Convolucions amb imatges en color

Per fer una convolució amb imatges en color es comença igual que quan es fa una convolució amb imatges en blanc i negre, triant un filtre. La diferència és que el filtre també ha de ser de tres dimensions. En ser de tres dimensions, s'obtenen tres filtres diferents, un per cada capa de color.

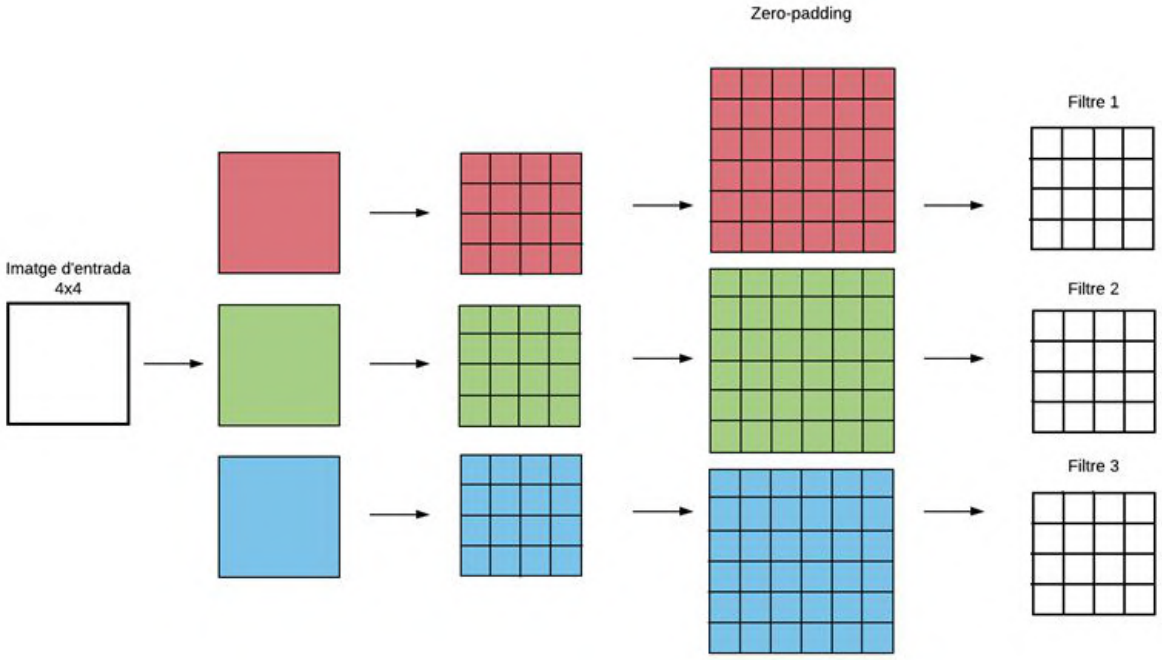


Figura 4.12: Representació de convolució en imatges RGB

Com es pot veure en la imatge 4.12, el procediment és el mateix, però en comptes de fer-ho amb una matriu, es fa amb tres.
Un cop aplicats els filtres, s'obtenen tres valors per cada píxel. Per obtenir una imatge nova, se sumen els tres resultats, obtenint un sol resultat per píxel, el qual se li suma el valor biaix, un paràmetre fix.

4.2.4 Max pooling amb imatges en color

En aquest cas, en comptes de tenir una entrada de dues dimensions és de tres, per tant, s'aplica el max pooling tres vegades, tal com es pot veure a la figura 4.13, obtenint així, una imatge de tres dimensions però de mida menor. Els passos per aplicar aquesta operació, són iguals que en una imatge en blanc i negre.

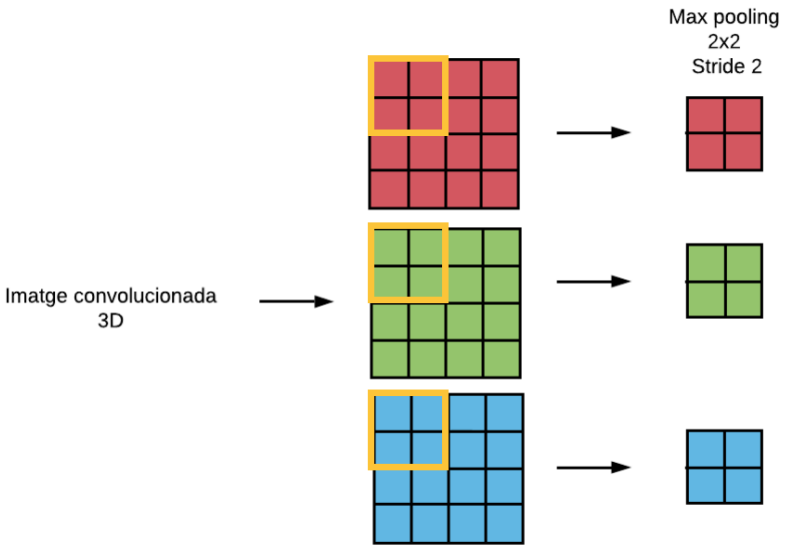


Figura 4.13: Representació de max pooling en imatges RGB

4.3 Entrenar el model

Una neurona o una xarxa neuronal no aprèn sola, necessita un aprenentatge o entrenament previ. En aquesta fase, la intel·ligència artificial ajustarà cada un dels pesos d'entrada de les neurones i els biaixos per tal que les respostes de la capa de sortida s'ajustin el màxim a les dades que coneixem.
Si recordem, els pesos són la importància que té cada connexió mentre que el biaix és un valor que fomenta que algunes neurones s'activin amb més facilitat que altres.

La imatge 4.14, mostra un exemple molt simplificat d'un entrenament, en aquest cas per detectar si la imatge pertany a un gos. La grossor de cada fletxa representa el pes de la connexió mentre que el valor final és l'exactitud, com major sigui, millor treballa la xarxa neuronal.

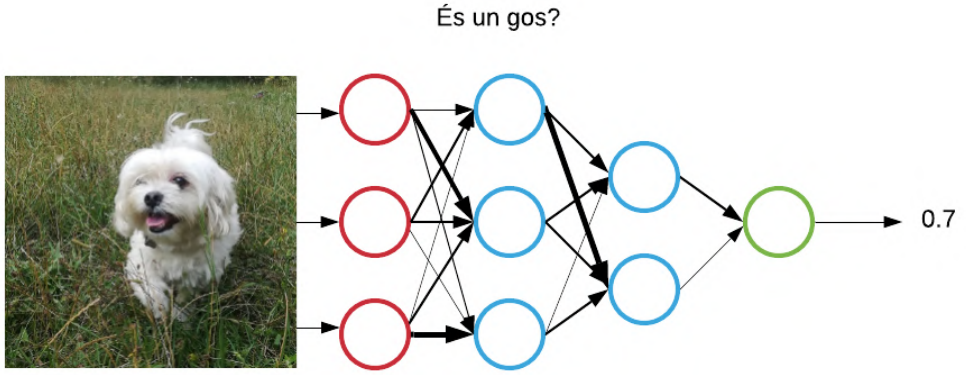


Figura 4.14: Exemple d'entrenament

A la imatge 4.15 es pot apreciar com canviant els pesos d'algunes connexions, s'obté una exactitud millor, aquesta és la manera com s'entrena una xarxa neuronal. Durant l'entrenament les connexions van variant els seus pesos, aquest canvi dels pesos produeix un canvi en l'exactitud. La xarxa neuronal observant aquesta variació d'exactitud, intentarà buscar els millors valors pels pesos per tal d'aconseguir l'estructura que treballi millor.

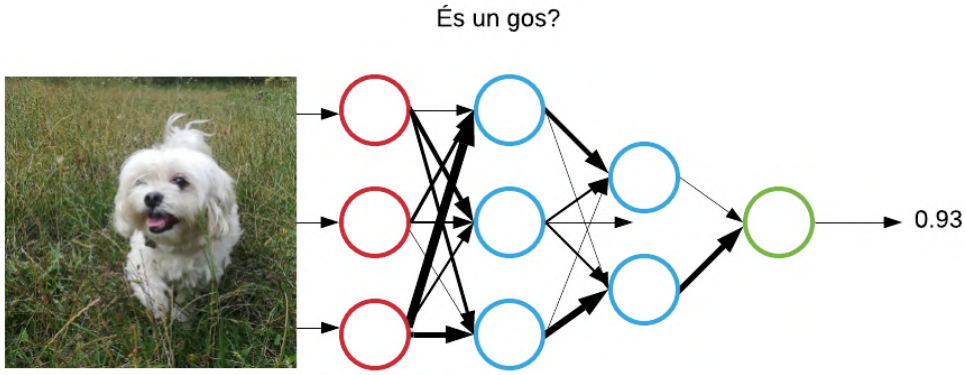


Figura 4.15: Exemple d'entrenament (II)

Si volem que la nostra intel·ligència artificial treballi correctament, és important utilitzar un elevat nombre de dades per entrenar. D'aquesta manera, la xarxa neuronal serà capaç d'ajustar els seus paràmetres per satisfer totes les imatges i per tant, augmentar el seu rendiment.

4.3.1 Tipus d'entrenament

La majoria de les tasques es basen en el coneixement adquirit mitjançant l'aprenentatge automàtic supervisat. És l'aprenentatge automàtic més habitual i fàcil de trobar, ja que és el més similar al que es coneix com a programació d'ordinadors.
Es parla d'aprenentatge supervisat quan es disposa d'un conjunt de dades amb les etiquetes[7] corresponents. La intel·ligència artificial analitzarà les dades buscant aquelles característiques que la puguin ajudar a calcular una sortida i l'intentarà buscar ella sola. Un cop trobada, la comprovarà amb la resposta entrada prèviament aprenent dels seus encerts i errors.
És a dir, si es busca que el model diferenciï fotos de gossos i gats, s'ha de facilitar una base de dades amb imatges d'aquests animals associades a una resposta o etiqueta que indiquin si el que hi ha a la imatge és un gos o un gat. S'anomena supervisat, ja que en entrar les respostes de l'algorisme, es diu que s'està supervisant l'aprenentatge.

Un altre aprenentatge que s'utilitza és l'aprenentatge no supervisat. Amb aquest aprenentatge s'aconsegueix que el model aprengui tan sols amb les dades d'entrada, sense necessitat d'explicar al sistema quin resultat es vol obtenir. En aquest cas, només es facilita a la intel·ligència els valors d'entrada, fent que els valors de sortida siguin una incògnita. El fet de no tenir referències prèvies dels valors de sortida, l'aprenentatge automàtic no supervisat utilitza la segmentació o agrupació d'elements similars, aplicant unes tècniques anomenades de clusterització.[8] En aquest aprenentatge no es busca una resposta única i concreta sinó que es busquen patrons de similitud de les dades d'entrada.

Per entendre millor la diferència entre els dos aprenentatges, fixem-nos en la imatge 4.16. En aquesta imatge s'hi pot veure un conjunt de dades format per quadrats i triangles, blaus o vermells. En el cas de l'aprenentatge supervisat, al model se li entren les dades d'entrada i les seves respectives respostes, és a dir, si la figura pertany a un quadrat o triangle i el seu color, de tal manera que quan se li entra una nova figura sabrà dir quin tipus de figura és i el seu color.
En el cas de l'aprenentatge no supervisat, la intel·ligència només té el conjunt de dades d'entrada. En aquest cas, com que la intel·ligència artificial no compta amb suficient informació per poder dir si l'objecte és un triangle o un quadrat i el seu color, busca patrons similars entre aquestes dades i les agrupa. És a dir, col·loca els quadrats en una fila i els triangles a una altra i a més a més els agrupa per colors. D'aquesta manera, si nosaltres entrem una nova dada, la intel·ligència artificial el posicionarà al grup que li pertoqui.
Gràcies a l'aprenentatge no supervisat, els programes són capaços d'entendre les normes d'un joc i per tant dissenyar estratègies per guanyar. També s'utilitza per crear les estratègies publicitàries, classificant a les persones en grups o en el reconeixement de veu, que amb el temps aprèn la manera de parlar d'una persona i les expressions que utilitza.

[7] Les etiquetes són el valor que el model ha de trobar, vindrien a ser les respostes correctes.
[8] Cluster o clustering en anglès.

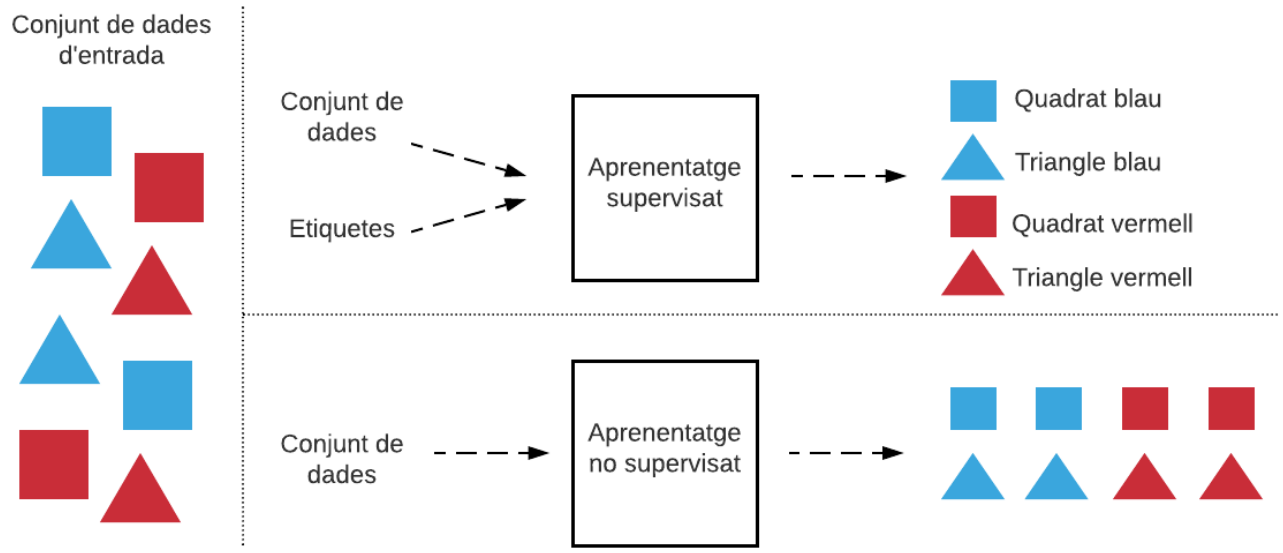


Figura 4.16: Tipus d'aprenentatge

4.4 Avaluar el model

Durant aquest pas s'observa com treballa el model utilitzant les dades destinades a l'avaluació.
Per visualitzar com treballa l'algorisme creat a través d'aprenentatge supervisat, s'acostuma a utilitzar una matriu de confusió, ja que permet veure quin tipus d'encerts i errors està tenint el model.
Per entendre-ho millor, es pot utilitzar un cas pràctic de la nostra realitat actual.
L'objectiu de la intel·ligència artificial de l'exemple següent, és crear un algorisme que ens permeti classificar pacients de COVID-19 en positius i negatius.
Les quatre opcions són les que conformen la matriu de confusió:[9]

1. Persona que té la COVID-19 i el model l'ha classificat com a positiu. S'anomena verdader positiu o VP.

2. Persona que no té la COVID-19 i el model l'ha classificat com a negatiu. És un verdader negatiu o VN.

3. Persona que té la COVID-19 i el model l'ha classificat com a negatiu. Això seria un error, i seria un fals negatiu o FN.

4. Persona que no té la COVID-19 i el model el classifica com a positiu. També és un error i s'anomena fals positiu o FP.

Per avaluar la qualitat del model es calculen 3 paràmetres, l'exactitud, la precisió i l'exhaustivitat.

[9]En aquest, com que només hi ha dues possibilitats, o ser positiu o negatiu, es tracta d'una matriu binària.

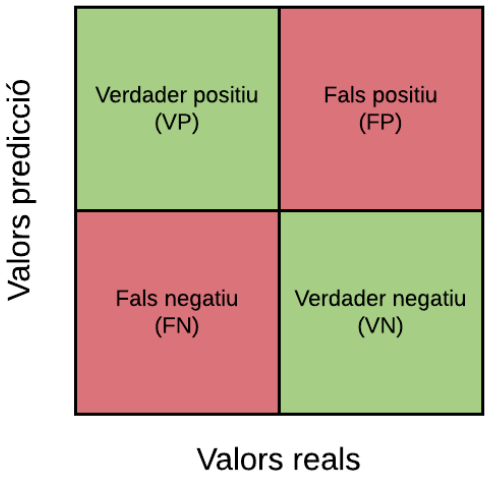


Figura 4.17: Matriu de confusió binària

L'exactitud es refereix a com de prop està el resultat al valor verdader. Es pot dir que l'exactitud és la quantitat de prediccions positives correctes.
Es representa com la proporció de resultats verdaders, tant positius com negatius, dividit entre el nombre total de casos examinats.

$$(VP + VN)/(VP + FP + FN + VN) \tag{4.1}$$

La precisió respon a la pregunta: *Quina proporció d'identificacions positives han sigut correctes?*
Es representa pels resultats verdaders positius dividits entre els resultats positius.

$$VP/(VP + FP) \tag{4.2}$$

Si el model té una precisió de 0.5, encerta el cinquanta per cent dels cops quan prediu un verdader.

L'exhaustivitat intenta respondre a la pregunta: *Quina proporció de positius reals s'han identificat correctament?*
Es calcula dividint els verdaders positius entre la suma de verdaders positius i falsos negatius.

$$VP/(VP + FN) \tag{4.3}$$

Per tant, si el model té una exhaustivitat de 0.11, identifica correctament un l'onze per cent dels positius.

4.4.1 Overfitting i underfitting

És molt comú que quan un crea una intel·ligència artificial, es trobi que no treballa de la manera desitjada. Això s'anomena overfitting i underfitting.
L'overfitting [10] passa quan el model amb les dades d'entrenament té una exactitud alta, però quan l'avaluem, aquesta exactitud és notablement més baixa.
Això passa quan el model ha memoritzat les dades en comptes de generalitzar i per tant buscar un algorisme. Per aquesta raó és important comptar amb dos conjunts de dades diferents, un per entrenar i l'altre per posar-lo a prova, ja que en cas contrari, no es podria veure com es comporta el model amb dades que no ha vist mai. També es considera un cas d'overfitting quan el nostre model intenta arribar a tal exactitud que no generalitza.
Aquest problema, pot passar per diferents motius, el més comú és per utilitzar una xarxa neuronal massa complexa o bé per entrenar massa el model.

L'underfitting[11] passa quan existeix un excés de generalització del model i ignora totes o la majoria de mostres de l'entrenament. Aquest cas és una conseqüència de no comptar amb les dades necessàries.
Si es crea un model per detectar els gossos d'una imatge, però les dades d'entrada només compta amb imatges de gossos d'una determinada raça, quan es mostri al model un gos d'una altra raça no mostrada durant l'entrenament, el model no ho reconeixerà com un gos. També es considera unferfitting si en el cas anterior, el model dictés que totes les imatges pertanyen a un gos malgrat que se li entri una imatge d'un cotxe, per exemple.

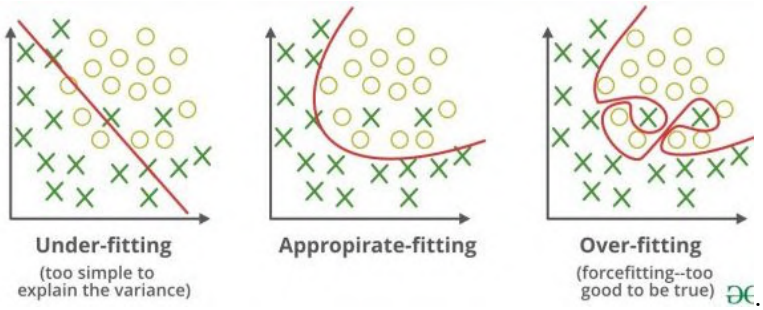


Figura 4.18: Overfitting i underfitting

A la imatge 4.18, es pot veure un model amb underfitting, que no compleix amb la seva funció, ja que no aconsegueix separar correctament les creus verdes amb els cercles. Seguidament, es pot observar un model amb un comportament apropiat, generalitza correctament, separant les creus verdes amb els cercles, tot i que com és normal, ho fa amb un marge d'error. Finalment, es pot observar un model amb overfitting, que separa les creus verdes i els cercles de manera tan exacta, que ja no fa una generalització, és a dir, que si nosaltres entrem un altre valor, segurament no ens el separarà correctament, ja que no ha creat unes normes generalitzades que serveixin per a la majoria dels casos.

[10] En català sobre ajustament.
[11] En català sobregeneralització.

4.4.2 Maneres d'evitar l'overfitting i l'underfitting

Com s'ha dit anteriorment, un problema pot ser que les dades d'entrada no siguin suficientment bones. Aquest problema es pot evitar utilitzant una tècnica anomenada augment d'imatges. Aquesta tècnica crea noves imatges d'entrenament aplicant un nombre aleatori de transformacions a les imatges que ja consten en el conjunt de dades. Un exemple d'aquesta tècnica és aplicar zoom, girar la imatge o aplicar l'efecte mirall. D'aquesta manera, s'aconsegueix tenir més dades d'entrada i fer que el model aprengui a generalitzar millor.
Una altra manera d'evitar els problemes d'overfitting, és el dropout. Un problema que podem tenir amb el nostre model és que un cop els pesos s'han modificat durant l'entrenament, una part de la xarxa tingui uns pesos molt elevats i l'altra part, molts baixos. Per tant, la part que té els pesos més elevats, cobrarà més importància mentre que l'altra part no influirà gaire en la decisió final.
El dropout, consisteix a canviar de manera aleatòria la xarxa neuronal, fent que algunes neurones deixin de funcionar durant una part de l'entrenament.
Fent això, es força a les altres neurones a participar i agafar més importància. Per tant a mesura que es duen a terme els epochs,[12] aconseguim que la xarxa es torni més resistent.

[12]El paràmetre anomenat epoch indica el nombre de vegades que es recorren les dades d'entrenament, mentre que el batch size indica el nombre de mostres que es processen durant cada epoch.

Capítol 5

Tecnologia utilitzada

A la imatge 5.1, s'hi pot veure un esquema de les tecnologies utilitzades per crear la intel·ligència artificial.

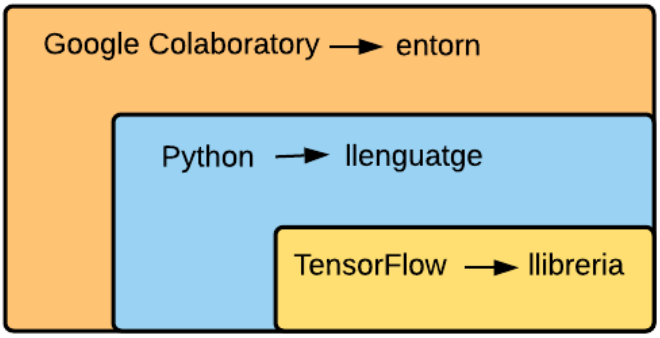


Figura 5.1: Esquema de les tecnologies utilitzades

5.1 Google Colaboratory

Google Colaboratory, també conegut com a Colab és un entorn de màquines virtuals. Un entorn és un programa informàtic que conté totes les eines i funcions necessàries per facilitar la programació. Els entorns permeten crear i editar projectes de desenvolupament, compilar un codi (traduir el programa a un codi que l'ordinador entengui, en aquest cas el binari), executar el codi i depurar el codi, és a dir, executar el codi pas per pas per tal de buscar el punt exacte on s'ha detectat l'error.
Els entorns estan formats per diferents components, com el llenguatge de programació, en aquest cas l'utilitzat és Python i unes llibreries, com és TensorFlow.

Google Colaboratory és un entorn destinat a la investigació de l'aprenentatge automàtic. És un servei gratuït molt útil que ofereix Google, en el qual es pot utilitzar cel·les de

text i cel·les de codi. Permet entrenar models d'aprenentatge automàtic de manera directa en el núvol, a més, no es necessita instal·lació i proporciona accés gratuït a una GPU.

5.1.1 GPU

Possiblement, l'accés a la GPU[1] és l'avantatge principal que ofereix Google Colaboratory. Una GPU és un hardware capaç de realitzar operacions matemàtiques sobre una gran quantitat de dades al mateix temps.
L'avantatge és que processa les dades en paral·lel, fent que encara que cada operació s'executi de forma més lenta del que ho fa la CPU, [2] el fet d'aplicar el paral·lelisme supera el rendiment de la CPU i permet obtenir les respostes més ràpidament.

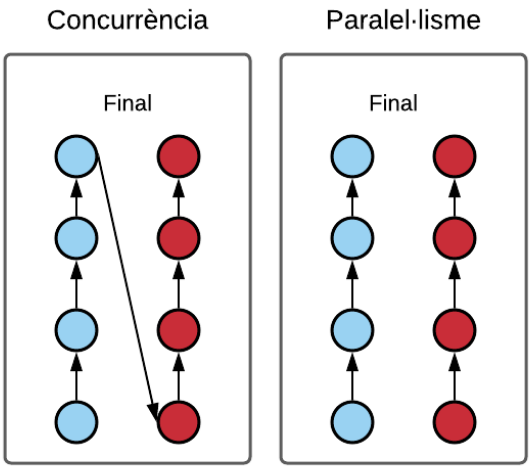


Figura 5.2: Representació de concurrència i paral·lelisme

A la figura 5.2, es pot veure la diferència entre la concurrència i el paral·lelisme. Un programa que aplica la concurrència, farà els processos d'un en un i seguint un ordre. Per exemple, si per cada operació de color blau necessita tres segons i les operacions de color vermell cinc segons, en el cas de la imatge en total necessitaríem 32 segons per dur a terme les operacions.
En el cas del paral·lelisme, els processos es distribueixen i es duen a terme simultàniament. En el cas de la imatge, mentre es duen a terme les operacions blaves, també es duen a terme les operacions vermelles, fent així que es tardin només 20 segons a fer les operacions, reduint el temps considerablement.

5.2 Python

Un entorn necessita un llenguatge de programació. En el cas de Google Colaboratory utilitza Python creat per Guido van Rossum l'any 1991.
Python és un dels llenguatges més populars a escala mundial i un dels més utilitzats a l'hora de programar intel·ligència artificial. Com a conseqüència s'ha convertit en el segon

[1] *Graphics Processing Unit.*
[2] *Central Processing Unit.*

llenguatge més utilitzat a GitHub.[3]
Un dels avantatges que presenta Python és la seva senzillesa i el seu aprenentatge relativament assequible.
Python està pensat per ser ampliable, és a dir, es compon per un nucli al qual se li poden afegir mòduls o llibreries. Moltes d'aquestes llibreries s'utilitzen per a la computació científica, la computació avançada i l'aprenentatge automàtic.

5.3 TensorFlow

La llibreria més important utilitzada en el projecte és TensorFlow. Una llibreria proporciona les funcions que el llenguatge de programació no té, en aquest cas, ens proporciona les funcions necessàries per a la construcció i entrenament de xarxes neuronals. És una llibreria desenvolupada per Google i actualment és una de les llibreries per xarxes neuronals més utilitzades en el món, utilitzada per companyies com Intel, Twitter, Ebay, Xiaomi, CocaCola, Google, entre d'altres.

TensorFlow rep aquest nom, ja que treballa amb dades representades en forma de tensor. Els tensors són estructures similars a les matrius o vectors. A més a més es diuen que aquestes estructures "flueixen"d'una neurona a una altra.

5.3.1 Tensors

Com s'ha dit anteriorment, els tensors són les estructures bàsiques utilitzades per TensorFlow; de fet cada operació produeix un tensor com a sortida. Un tensor és un vector o matriu multidimensional que representa les dades del programa. Dit amb altres paraules, la xarxa neuronal opera amb tensors, que són les dades en forma de vector o matriu. A la imatge 5.3, es poden veure els tensors més bàsics. El primer que s'hi pot veure és un escalar format per un únic nombre. Seguidament, s'hi pot veure un vector, similar a una llista de valors i finalment una matriu.
En el processament d'imatges, les matrius són les estructures més utilitzades. En el cas de la imatge, la matriu té dos eixos. Aquestes matrius ens ajuden a tractar imatges en blanc i negre, on un eix pertany a l'altura i l'altre eix a l'amplada, en aquest cas l'eix que val 2 equival a l'amplada mentre que el que val 3 equival a l'altura.

[3]Github és un repositori en línia gratuït que permet gestionar projectes. És molt utilitzat pels desenvolupadors per guardar els seus treballs donant així l'oportunitat a milions de persones de tot el món a cooperar amb ells.

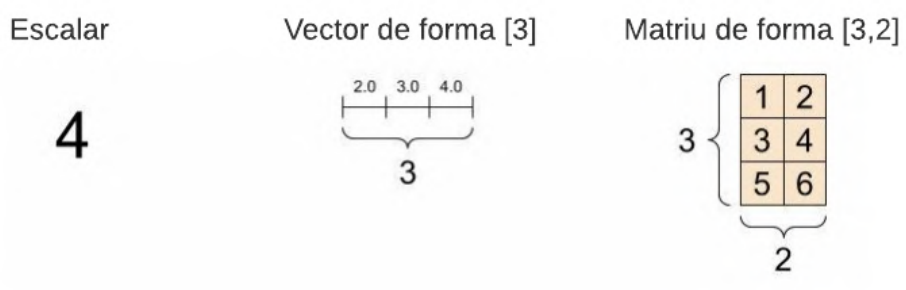


Figura 5.3: Representació de tensors bàsics

Les imatges en color en canvi, es representen generalment mitjançant tres valors, l'amplada, l'altura i la profunditat. Aquestes imatges es codifiquen a partir de tensors 3D, és a dir, matrius de tres eixos com la de la imatge 5.4.

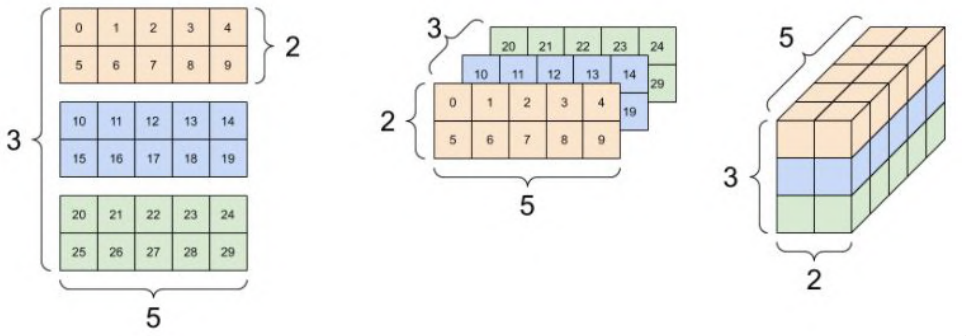


Figura 5.4: Representació d'una matriu de forma [3,2,5]

```
[[[[252   2   1]
   [248   0   1]
   [249   0  10]
   ...
   [248   0   4]
   [252   4   3]
   [246   0   0]]

  [[249   1   1]
   [234   1   3]
   [209   6  15]
```

Figura 5.5: Representació d'una imatge

A la imatge 5.5 es pot veure una imatge en color representada a partir d'un tensor de tres dimensions.

Capítol 6

Crear una intel·ligència artificial

Un cop explicats els conceptes bàsics, l'objectiu principal es va enfocar en crear una intel·ligència artificial des de zero. D'aquesta manera, els conceptes explicats en els capítols anteriors podien ser aplicats en un problema real. L'objectiu d'aquest capítol és explicar la manera com es va desenvolupar aquesta intel·ligència artificial.
Es va començar esquematitzant els passos del procés.[1] Aquests passos són els explicats anteriorment. El procés comença definint els objectius de la intel·ligència artificial, un cop definits, s'ha d'escollir el conjunt de dades per poder entrenar el model i s'han de preparar aquestes dades per tal que la intel·ligència artificial les pugui processar. Seguidament s'han de crear la xarxa neuronal que s'entrena utilitzant les dades d'entrada. Un cop acabat l'entrenament, la intel·ligència artificial torna el valor de l'exactitud que indica el percentatge d'encerts del model. Si aquesta exactitud és més baixa de l'esperada, es pot canviar l'estructura de la xarxa neuronal o escollir un altre conjunt de dades. Si, per altra banda, l'exactitud és satisfactòria, s'ha de guardar el model i si es desitja, implementar-lo en una aplicació.

6.1 Definir objectius

Per començar calia definir els objectius, és a dir, decidir quin tipus de problema la intel·ligència artificial havia de resoldre. Des del principi la intenció era resoldre un problema de classificació d'imatges que tingués com a finalitat poder ajudar a la gent.
Per agafar idees es van buscar exemples d'intel·ligències artificials relacionades amb la classificació d'imatge. N'hi havia que detectaven les emocions o fins i tot si la persona portava la mascareta o no i si la portava ben posada. No obstant això, cap tenia la finalitat desitjada. Per tant es va canviar l'enfocament i centrar la recerca amb conjunts de dades ja creats (és a dir, dades d'entrada). Cal aclarir que per aconseguir les dades d'entrada d'una intel·ligència artificial, es pot fer de dues formes: crear les imatges un mateix, fet que et garanteix les dades siguin més adequades al problema, però una quantitat molt gran de temps i de feina, o bé es pot utilitzar un conjunt de dades ja creat, és a dir, les dades que alguna persona ja va agrupar per resoldre un problema determinat.

[1] Vegeu l'esquema a l'annex 1: Passos seguits.

Un dels conjunts de dades analitzats estava basat en l'abecedari ASL,[2] un llenguatge visual format per gestos fets amb una mà, utilitzat sobretot als Estats Units i el Canadà.[3] Aquest conjunt de dades va servir per poder determinar l'objectiu final de la intel·ligència artificial, traduir a text les lletres de l'abecedari ASL.

6.2 Escollir el conjunt de dades d'entrada

Un cop decidit l'objectiu de la intel·ligència artificial, es va buscar un conjunt de dades, relacionat amb l'abecedari ASL, que podria anar bé per entrenar el model.

El primer conjunt utilitzat estava format per 2928 imatges.[4] Era un model que no comptava amb gaires dades. Com a conseqüència, malgrat canviar l'estructura de la xarxa neuronal, no s'aconseguia una exactitud superior al 50%; a més, el model no generalitzava correctament, és a dir, patia d'underfitting.
Per saber que un model pateix d'underfitting, s'han d'analitzar les exactituds tant de l'entrenament com de l'avaluació.

```
Epoch 1/5
91/91 [==============================] - 15s 156ms/step - loss: 3.1433 - accuracy: 0.2803
Epoch 2/5
91/91 [==============================] - 14s 154ms/step - loss: 2.9331 - accuracy: 0.4928
Epoch 3/5
91/91 [==============================] - 14s 156ms/step - loss: 2.8684 - accuracy: 0.5569
Epoch 4/5
91/91 [==============================] - 14s 154ms/step - loss: 2.8383 - accuracy: 0.5872
Epoch 5/5
91/91 [==============================] - 14s 155ms/step - loss: 2.8278 - accuracy: 0.5976
```

Figura 6.1: Entrenament

A la imatge 6.1, es pot veure l'entrenament dut a terme amb aquestes dades d'entrada. En aquest cas, només són importants els valors sota el nom d'*accuracy*, en català exactitud. En aquest cas, l'exactitud més alta és de 0.5976 o 59.76%.

```
model.evaluate(test_images, test_labels)
```

```
1/1 [==============================] - 0s 203ms/step - loss: 2.9601 - accuracy: 0.4643
[2.9601352214813232, 0.4642857015132904]
```

Figura 6.2: Avaluació

[2]American Sign Language.
[3]Per veure els signes vegeu l'annex 2: Abecedari ASL.
[4]Es pot trobar a https://www.kaggle.com/lcastrillon/mini-asl-alphabet#link.

En el cas de la imatge 6.2, es pot veure l'avaluació de la mateixa xarxa neuronal. L'exactitud és de 0.4643 o del 46.43%. Si es compara amb l'exactitud de l'entrenament, es pot apreciar com aquesta és notablement més baixa, el que ens indica que el model no generalitza correctament. Aquest problema es va associar a la falta de dades.
Aquest exemple serveix per demostrar que les dades són realment importants, no només la qualitat sinó que també la quantitat. En aquest cas les imatges eren clares i adequades però no suficients per a la complexitat de la tasca.

Quan no s'aconsegueix l'exactitud esperada, tal com s'ha dit anteriorment, hi ha dues opcions, o bé canviar la xarxa neuronal o bé canviar el conjunt de dades. En aquest cas, com que el problema es trobava en el conjunt de dades, per més que es canviava l'estructura de la xarxa neuronal no s'aconseguia augmentar l'exactitud, per tant, es va decidir canviar de conjunt de dades.

El segon conjunt de dades era un conjunt de dades format per 52.000 imatges.[5]
Només en canviar el conjunt de dades, sense variar l'estructura de la xarxa neuronal, l'exactitud va augmentar significativament tal com es pot veure a les imatges 6.3 i 6.4.

```
Epoch 1/5
1422/1422 [==============================] - 228s 160ms/step - loss: 2.6066 - accuracy: 0.7166
Epoch 2/5
1422/1422 [==============================] - 224s 158ms/step - loss: 2.4789 - accuracy: 0.8428
Epoch 3/5
1422/1422 [==============================] - 226s 159ms/step - loss: 2.4495 - accuracy: 0.8725
Epoch 4/5
1422/1422 [==============================] - 223s 157ms/step - loss: 2.4473 - accuracy: 0.8745
Epoch 5/5
1422/1422 [==============================] - 223s 157ms/step - loss: 2.4474 - accuracy: 0.8745
```

Figura 6.3: Entrenament

En aquest cas tornava a haver-hi una diferència entre l'exactitud de l'entrenament i l'avaluació, no obstant això, com que era una diferència mínima no era important.

```
model.evaluate(test_images, test_labels)

204/204 [==============================] - 20s 100ms/step - loss: 2.4674 - accuracy: 0.8545
[2.467362880706787, 0.8544615507125854]
```

Figura 6.4: Avaluació

Un cop vist aquest canvi, es va decidir continuar millorant el model canviant l'estructura de la xarxa neuronal. Finalment, després de provar diferents xarxes neuronals, es va aconseguir una exactitud màxima de 0.997 en l'entrenament i 0.9802 en l'avaluació, el que indicava que el model treballava bé en un 98% dels casos.

[5]Es pot trobar a https://www.kaggle.com/rolandomr/american-sign-language-recogntion-asl#link.

Aquest model semblava que treballava suficientment bé. El problema va sorgir quan se li va ordenar que analitzés una imatge que no havia vist ni durant l'entrenament ni durant l'avaluació. En aquest cas, el model no aconseguia predir bé cap de les imatges. El problema tornava a radicar en el conjunt de dades, es tenien moltes imatges, però la qualitat d'aquestes no era suficientment alta.
A la imatge 6.5, es poden veure tres exemples d'imatges que formen part del segon conjunt de dades utilitzat. La primera imatge pertany a la lletra A, la segona a la lletra E i la tercera a la lletra M. Són tres lletres diferents, però que per culpa de l'estil de la imatge, les diferències són poc notables o inexistents.

(a) Lletra A (b) Lletra E (c) Lletra M

Figura 6.5: Exemples d'imatges

El fet que totes les imatges es presentessin amb un fons negre i la silueta de la mà de color blanc, va fer que quan s'entrava qualsevol imatge on apareixien dits o simplement un fons diferent, la intel·ligència artificial no la sabés classificar. Aquest cas és un clar exemple que tenir moltes imatges per entrenar, no serveix si aquestes no són suficientment bones.

Es va decidir utilitzar un tercer conjunt de dades, aquesta vegada es buscava un conjunt amb suficients dades que fossin de qualitat. Malgrat tot, no es va aconseguir trobar un conjunt de dades que satisfés les necessitats, ja que eren conjunts amb masses dades i com a conseqüència, l'ordinador utilitzat per correr la intel·ligència artificial no les podia processar.
Finalment es va decidir crear un conjunt de dades des de zero, així es podia crear un conjunt amb una quantitat de dades que l'ordinador pogués processar i suficientment bones per poder entrenar el model. Per fer-ho es van recol·lectar imatges de totes les lletres. Per l'organització de les imatges es va imitar l'ordre dels altres conjunts de dades utilitzats. La carpeta del conjunt de dades estava formada per dues carpetes, la que contenia les dades de l'entrenament i la que contenia les dades de l'avaluació. Cada una de les dues carpetes tenia 26 carpetes en el seu interior una per cada lletra, i dins d'aquestes carpetes, s'hi van guardar les imatges. Cada carpeta tenia el nom de la lletra que s'hi representava en les imatges del seu interior. En total, el conjunt estava format per 50.000 imatges.

En aquest cas, la màxima exactitud que es va aconseguir va ser del 0.8082 en el cas de l'entrenament i 0.7891 en el cas de l'avaluació. Encara que l'exactitud no fossin tan alta com en el cas del conjunt de dades anteriors, la intel·ligència artificial era capaç de poder predir de forma satisfactòria imatges no vistes ni en l'entrenament ni en l'avaluació.

6.3 Preparar de les dades

Com s'ha anteriorment, el model treballa amb valors numèrics, per tant, les imatges s'han de transformar en tensors.[6]
Com que l'entrenament que es volia dur a terme era supervisat, el model necessitava unes imatges d'entrada i les seves respectives respostes per tal d'aprendre. Tenir les imatges agrupades en carpetes va facilitar a l'hora de crear el conjunt de respostes al mateix temps que convertia les imatges a tensors. Per fer-ho, el programa agafava una imatge, li canviava la mida per tal de tenir totes les imatges amb el mateix nombre de píxels i les convertia a matrius. Al mateix temps, agafava el nom de la carpeta on estava situada la imatge, que corresponia a la lletra representada a la imatge i l'afegia a la llista de respostes.
Seguidament, transformava les respostes en valors numèrics perquè la intel·ligència artificial pogués entendre-les i es reduïa el valor dels píxels a un nombre entre 0 i 1, ja que així els models treballen millor.

6.4 Crear de la xarxa neuronal

Un cop les dades ja tenien el format necessari, es va procedir a crear la xarxa neuronal. Com que no hi ha unes dades generals per aplicar a l'hora de crear una xarxa neuronal, per determinar la millor estructura, es van provar moltes estructures diferents i analitzant l'exactitud, es va determinar la xarxa que treballava millor.

La primera xarxa neuronal que es va provar es tractava d'una xarxa neuronal típica,[7] tal com es pot veure a la imatge 6.6.

```
[ ]  model= tf.keras.Sequential([
       tf.keras.layers.Flatten(input_shape=(200,200)),
       tf.keras.layers.Dense(128,activation=tf.nn.relu),
       tf.keras.layers.Dense(29, activation= tf.nn.softmax)
     ])
```

Figura 6.6: Primera xarxa neuronal

[6] Vegeu el codi d'aquesta part a l'annex 3: Preparar les dades.

[7] Les xarxes neuronals típiques són xarxes formades per capes no especialitzades en cap procés en concret.

Amb aquesta xarxa neuronal es va aconseguir només una exactitud de 0.0357, el que indica que la xarxa neuronal només encertava un 3,57% dels casos.
Com que l'exactitud no era l'esperada es va decidir canviar l'estructura de la xarxa neuronal, aquesta vegada, aplicant una xarxa neuronal convolucional.
Després de provar diverses estructures, és a dir, després d'afegir capes d'amagades i variar el nombre de neurones mentre s'estudiava l'exactitud de la xarxa per saber si aquesta millorava o empitjorava, es va arribar a la xarxa neuronal amb la qual es va aconseguir l'exactitud més gran,[8] del 98% amb el segon conjunt de dades i del 78.91% amb l'últim conjunt de dades.
Per entendre l'estructura, es divideix la xarxa neuronal en dues parts, tal com es pot veure a la imatge 6.7.

```
model= tf.keras.Sequential([
  tf.keras.layers.Conv2D(64,(3,3), activation=tf.nn.relu, input_shape=(64,64,3)),
  tf.keras.layers.MaxPooling2D((2,2), strides=2),

  tf.keras.layers.Conv2D(64,(3,3), activation=tf.nn.relu),
  tf.keras.layers.MaxPooling2D((2,2), strides=2),

   tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(64,activation=tf.nn.relu),
  tf.keras.layers.Dense(26,activation=tf.nn.softmax)
])
```

.

Figura 6.7: Xarxa neuronal

La part blava està formada per capes pròpies d'una xarxa neuronal convolucional, és a dir, capes de convolucions, anomenades *Conv2D* i capes de MaxPooling, anomenades *MaxPooling2D*, mentre que l'altra part està formada per capes pròpies d'una xarxa neuronal tradicional.

A la part blava es pot veure com un cop les dades tenen el format necessari per poder treballar amb elles, es processen a la primera capa de la xarxa neuronal, en aquest cas una capa convolucional on s'aplica un filtre 3x3 que ajuda a detectar certes característiques. Seguidament, s'aplica a les dades la funció MaxPooling, amb una taula de 2x2 i un stride de 2. Aquests dos passos, es repeteix abans de passar a les capes tradicionals.

El fet de tenir més d'una capa convolucional, ajuda a detectar estructures més complexes. Tal com es pot veure a la figura 6.8, com més profundes són les capes, són capaces de detectar més estructures.

[8] Vegeu l'annex 4: Xarxa neuronal.

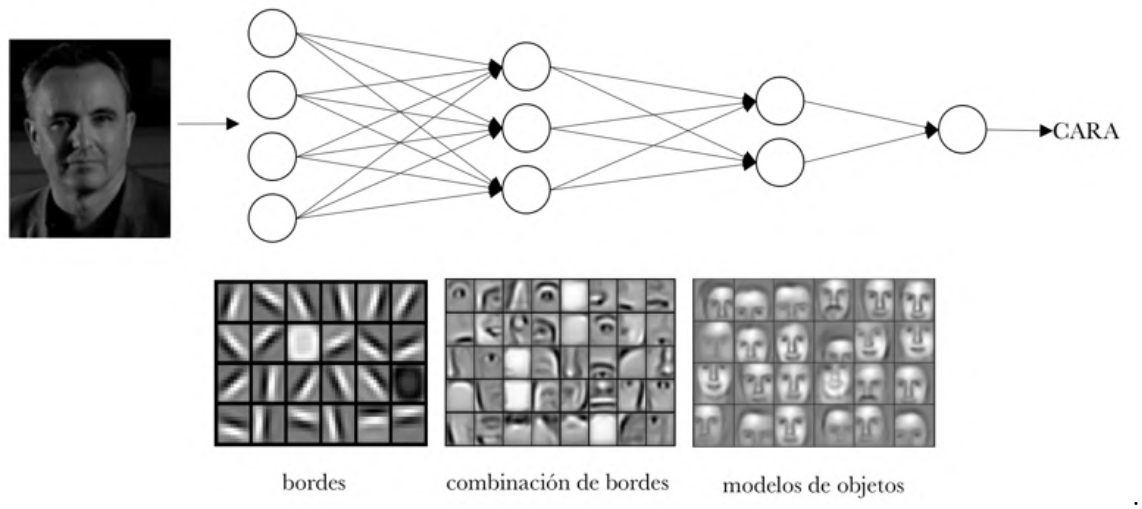


Figura 6.8: Xarxes convolucionals

Un cop aplicades aquestes capes, s'ha de predir la imatge. Per fer-ho, s'utilitza capes pròpies d'un programa clàssic o tradicional, aquestes capes són les capes *Flatten* i les capes *Dense*. Aquestes capes serveixen per agrupar la informació i extreure'n una conclusió, en aquest cas, referent a la lletra representada.
L'última capa està formada per tantes neurones com classes diferents hi ha al problema, en aquest cas, les classes són les lletres, per això que està formada per 26 neurones. Aquesta capa torna 26 valors que indiquen la probabilitat que la imatge pertanyi a cada lletra, és a dir, per una imatge la intel·ligència torna el valor de la probabilitat que pertanyi a la lletra A, el valor de la probabilitat que pertanyi a la lletra B, la probabilitat que pertanyi a la lletra C...I així successivament per totes les lletres. Com a resultat del problema, s'agafa la probabilitat més elevada.

Millorar la xarxa neuronal va ser el procés al qual se li va dedicar més temps. En total es van provar més de trenta xarxes neuronals, variant tant el nombre de capes, com el nombre de neurones de cada capa.

La xarxa neuronal de la figura 6.9, és la xarxa neuronal amb més capes i més neurones, però que tan sols tenia una exactitud de 0.346. Aquesta estructura mostra com una capa més complexa, no és sinònim de major exactitud.

```
model= tf.keras.Sequential([
  tf.keras.layers.Conv2D(32,(3,3), activation=tf.nn.relu, input_shape=(64,64,3)),
  tf.keras.layers.MaxPooling2D((2,2), strides=2),

  tf.keras.layers.Conv2D(64,(3,3), activation=tf.nn.relu),
  tf.keras.layers.MaxPooling2D((2,2), strides=2),

  tf.keras.layers.Conv2D(128,(3,3), activation=tf.nn.relu),
  tf.keras.layers.Conv2D(128,(3,3), activation=tf.nn.relu),

  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(256,activation=tf.nn.relu),
  tf.keras.layers.Dense(26,activation=tf.nn.softmax)
])
```

Figura 6.9: Xarxa neuronal més complexa

6.5 Entrenar la xarxa neuronal

El següent pas va ser entrenar el model.[9] Com que es volia utilitzar l'entrenament supervisat, per entrenar el model es va utilitzar la funció *model.fit* i com a dades d'entrada, les imatges d'entrenament i les seves respectives respostes. Seguidament es van haver de determinar dos valors, els *epochs* i el *batch*. El valor dels epochs determina les vegades que les imatges són analitzades per la xarxa neuronal, és a dir, en el cas del model, que té un valor de 100 epochs, les imatges d'entrada seran analitzades 100 vegades. El problema és que la quantitat de dades que la xarxa neuronal ha d'analitzar és massa gran perquè la intel·ligència artificial les analitzi totes de cop, per això s'empra el valor de *batch*. Per entrar les dades aquestes es divideixen. El valor del batch indica quantes imatges es volen en cada divisió.
Per entendre-ho millor, imaginem-nos que hem de firmar una quantitat de fulls. Com que la quantitat de fulls que hem de firmar és tan elevada, repartim els fulls en diferents calaixos, de tal manera que començarem firmant un calaix i quan tinguem tots els fulls firmats d'aquell calaix, passarem a firmar els fulls d'un altre calaix fins a tenir tots els fulls firmats. En el cas de la intel·ligència artificial, cada full per firmar és una imatge a analitzar i la quantitat de fulls que posem a cada calaix és el valor *batch*. Si cada full s'hagués de firmar 5 vegades, es repetiria el mateix procediment 5 vegades. En aquest cas, el nombre de vegades és el que s'anomena *epoch*.

Durant l'entrenament, es va estar modificant tan sols el valor dels epochs, ja que era amb el que es va notar més diferència. En les següents imatges es pot veure com varia l'exactitud variant tan sols el nombre d'epochs.

[9] Vegeu l'annex 5: Entrenar el model.

```
Epoch 25/30
1219/1219 [==============================] - 304s 249ms/step - loss: 2.6905 - accuracy: 0.6317
Epoch 26/30
1219/1219 [==============================] - 304s 249ms/step - loss: 2.6843 - accuracy: 0.6378
Epoch 27/30
1219/1219 [==============================] - 304s 249ms/step - loss: 2.6748 - accuracy: 0.6476
Epoch 28/30
1219/1219 [==============================] - 304s 249ms/step - loss: 2.6687 - accuracy: 0.6537
Epoch 29/30
1219/1219 [==============================] - 304s 249ms/step - loss: 2.6614 - accuracy: 0.6605
Epoch 30/30
1219/1219 [==============================] - 304s 249ms/step - loss: 2.6579 - accuracy: 0.6642
```

Figura 6.10: Entrenament 30 epochs

A la imatge 6.10, es pot veure una part de l'entrenament. Al costat esquerre hi ha el nombre d'epochs, en aquest cas, tan sols es veuen els 5 últims epochs de l'entrenament. A l'altra banda dels guions, hi ha indicat els segons s'ha tardat a realitzar cada epoch i els segons que s'han tardat a realitzar cada pas o operació. Seguidament hi ha valor de pèrdua del model i l'exactitud. En aquest cas, l'exactitud és del 66.42%.

A la imatge 6.11, podem veure la mateixa xarxa neuronal però amb el doble d'epochs.

```
Epoch 55/60
1422/1422 [==============================] - 18s 13ms/step - loss: 2.5176 - accuracy: 0.8044
Epoch 56/60
1422/1422 [==============================] - 18s 13ms/step - loss: 2.5157 - accuracy: 0.8062
Epoch 57/60
1422/1422 [==============================] - 18s 12ms/step - loss: 2.5154 - accuracy: 0.8065
Epoch 58/60
1422/1422 [==============================] - 18s 13ms/step - loss: 2.5146 - accuracy: 0.8074
Epoch 59/60
1422/1422 [==============================] - 18s 12ms/step - loss: 2.5141 - accuracy: 0.8078
Epoch 60/60
1422/1422 [==============================] - 18s 13ms/step - loss: 2.5137 - accuracy: 0.8082
```

Figura 6.11: Entrenament 60 epochs

En aquest cas l'exactitud va augmentar fins al 80.82%. Això es deu al fet que durant cada epoch, la intel·ligència artificial va agafant experiència i per tant, va aprenent, fent per tant que sàpiga resoldre millor els problemes un cop ja porta uns cicles aprenent.
L'entrenament va ser el pas que necessitava més estona, ja que era un procés molt lent. El temps que tardava la intel·ligència a entrenar-se depenia tant de la quantitat de dades d'entrada, com de l'estructura de la xarxa neuronal i dels valors dels epochs i els batch. Com a mitjana realitzava aproximadament entre set i deu epochs cada hora.

6.6 Avaluar i guardar el model

Un cop entrenat el model es va posar a prova.[10] Per fer-ho es va utilitzar la funció *model.evaluate* juntament amb el conjunt d'imatges destinades a l'avaluació i les seves respectives respostes. D'aquesta manera el nostre model provava de predir la solució de les imatges del conjunt i comparava les seves prediccions amb les respostes entrades per tal de calcular l'exactitud.

```
model.evaluate(test_images, test_labels)

213/213 [==============================] - 2s 7ms/step - loss: 2.5331 - accuracy: 0.7891
[2.5330872535705566, 0.7890946269035339]
```

Figura 6.12: Avaluació

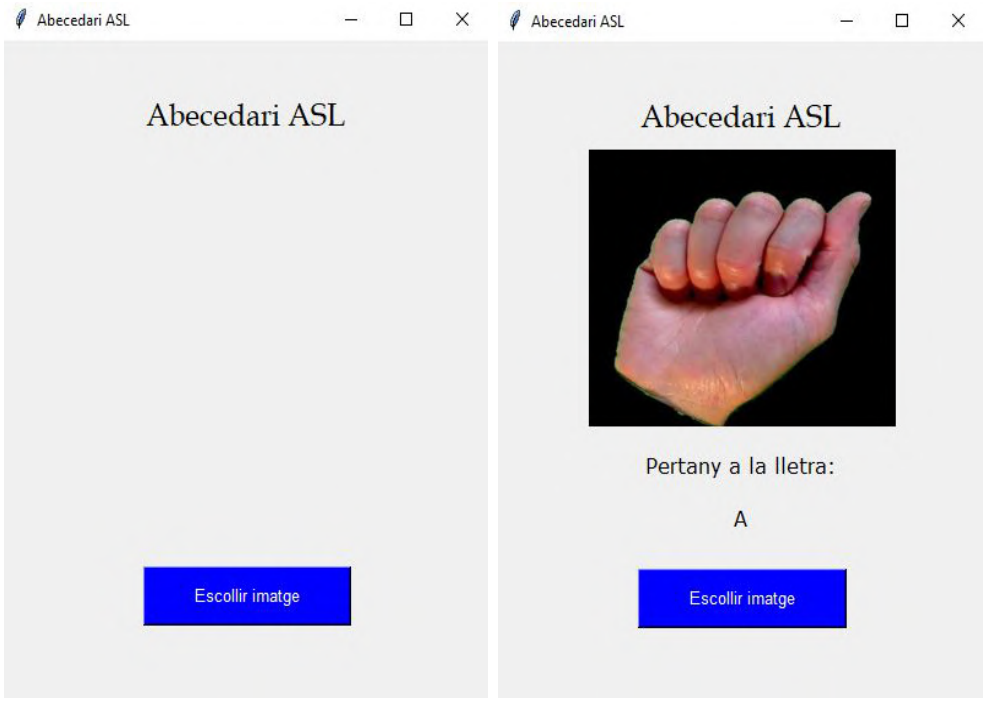
A la imatge 6.12 es pot observar l'avaluació del model final, amb una exactitud de 0.7891. Seguidament es va guardar el model, és a dir, guardar el valor dels pesos de cada connexió. Si no es guarda el model, cada vegada que es vulgui utilitzar, la intel·ligència artificial s'haurà de tornar a entrenar, fet que pot tardar diverses hores.

6.7 Aplicar el model

Per poder mostrar com treballa la intel·ligència artificial es va crear una aplicació amb Python.[11]

A la figura 6.13, es poden veure dues captures de l'aplicació. La imatge (a) és l'inici de l'aplicació on hi ha un botó que permet escollir la imatge. La imatge (b), és la sortida del programa on es mostra la imatge que s'ha seleccionat i a sota la predicció que ha fet la intel·ligència artificial, en aquest cas, la imatge pertany a la lletra A.
Un cop prenem el botó i escollim la imatge, s'activa una funció que per una banda ens mostra la imatge mentre que per l'altre converteix la imatge en un tensor per tal que la intel·ligència la pugui interpretar. Seguidament s'importa al programa la intel·ligència artificial i se li ordena que faci una predicció amb la funció *model.predict*.

[10] Vegeu l'annex 6: Avaluar i guardar el model.
[11] Vegeu l'annex 7: Aplicar el model.



(a) Seleccionar l'entrada (b) Sortida

Figura 6.13: Aplicació

A la imatge 6.14, es pot veure la predicció de la intel·ligència artificial per aquest mateix cas.

```
[[9.99998689e-01 0.00000000e+00 0.00000000e+00 0.00000000e+00
  1.47176535e-36 0.00000000e+00 0.00000000e+00 1.64214648e-16
  6.11251047e-34 0.00000000e+00 2.57060132e-38 1.13589199e-13
  0.00000000e+00 1.59863440e-29 6.88703488e-37 4.51034578e-36
  9.28896475e-30 0.00000000e+00 5.29528458e-20 1.01805965e-20
  3.48308780e-31 2.64424839e-32 0.00000000e+00 2.34926264e-23
  1.25769157e-06 0.00000000e+00]]
0
```

Figura 6.14: Abecedari ASL

Com s'ha dit anteriorment, quan la intel·ligència fa una predicció retorna 26 valors, cada valor correspon a la probabilitat que la imatge pertanyi a una lletra, d'entre tots els valors, es busca el més elevat, serà la solució al problema. Finalment, només cal transcriure la posició de la probabilitat més elevada a la lletra de l'abecedari que correspon. En aquest cas com que el valor més elevat és el primer, és a dir, el de la posició 0, la imatge pertany a la lletra A.

Podeu trobar el codi de l'aplicació a GitHub a través de l'enllaç:
`https://github.com/nuriallfe/Intel-lig-ncia-artificial.git`.

Capítol 7

Conclusions

L'objectiu inicial d'aquest treball era comprendre què és la intel·ligència artificial i entendre com treballa, no obstant això, més endavant aquest objectiu es va enfocar en crear una aplicació amb intel·ligència artificial. Un cop fet el treball i aconseguits els objectius acabats de mencionar, s'ha pogut arribar a diverses conclusions.

Fent aquest treball s'ha pogut observar com els primers passos van ser crucials pel desenvolupament de l'aplicació. És a dir, dedicar un temps a comprendre els conceptes més importants que envolten la intel·ligència artificial, ha sigut la base de tot el treball.

Per altra banda, el fet de cometre dos erros a l'hora d'escollir el conjunt de dades d'entrada, ha permès veure la importància que aquestes tenen en la intel·ligència artificial. En total s'han utilitzat tres conjunts de dades. El primer ha servit per mostrar que la quantitat de dades d'entrada ha de ser proporcional a la complexitat del problema. Per altra banda, el segon conjunt de dades, ha servit per demostrar que no només és important la quantitat de dades sinó que també la qualitat d'aquestes. A més a més, el fet de poder crear el conjunt de dades des de zero, també ha servit per poder diferenciar quines imatges són adequades per un problema i quines no.

Durant el procés per trobar la millor estructura de la xarxa neuronal, s'ha pogut comprovar com aquest és un procés llarg i que necessita temps. S'ha pogut veure com una xarxa neuronal complexa no sempre és una bona opció, tot i que una xarxa neuronal massa senzilla, és a dir, amb poques neurones i/o capes d'amagades, no sempre funciona com hauria sinó que cada problema necessita una xarxa neuronal diferent. Per altra banda, comparant l'exactitud d'una xarxa neuronal típica amb la d'una xarxa neuronal convolucional s'ha pogut apreciar els avantatges d'utilitzar una xarxa neuronal convolucional per resoldre problemes de classificació d'imatges, ja que en canviar d'una xarxa neuronal típica a una xarxa neuronal convolucional, l'exactitut ha augmentat significativament.

Finalment, la conclusió més clara d'aquest treball recalca la importància de la intel·ligència artificial està agafant a la nostra societat i com pot canviar el nostre futur i la manera de programar. En aquest treball, s'ha pogut desenvolupar una intel·ligència artificial que

amb poques línies de text, i pràcticament cap indicació ha estat capaç d'aprendre a distingir entre 26 signes de l'abecedari ASL, una tasca que amb programació tradicional hauria necessitat moltes condicions i línies de text.

Cal assenyalar que aquest treball presenta algunes limitacions, sobretot en l'àmbit informàtic. Durant el procés de desenvolupament de la intel·ligència artificial van sorgir diversos problemes relacionats amb l'entorn Google Colaboratory. Malgrat tenir accés a la GPU de l'entorn, el temps que tardava cada xarxa neuronal a entrenar-se eren molt extens. A més a més, l'entorn es desconnectava amb freqüència fent que hagués de tornar a començar tot el procés des d'importar les dades fins a l'entrenament. Un altre problema detectat va ser relacionat amb la RAM,[1] com que no tenia suficient espai, no podia emmagatzemar una gran quantitat de dades. Aquest fet va comportar que no es pogués fer ús de diversos conjunts de dades per la seva mida. Un cop analitzats aquests problemes, si s'hagués de tornar a fer el treball, es contractaria el servei de pagament mensual de Google Colaboratory Pro, que garanteix més CPU, augment de la RAM i que es desconnecti amb menor freqüència.

A l'hora de programar l'aplicació, també van sorgir problemes per importar certes llibreries, fet que es va poder arreglar creant un nou entorn d'execució amb Anaconda. També es vol destacar que l'escrit del text ha estat desenvolupat a partir de LaTeX, un llenguatge que ha comportat diversos problemes al llarg del procés. A més a més, gairebé totes les imatges i esquemes utilitzats per l'escrit, han sigut creats des de zero.

L'exactitud de la intel·ligència creada és del 0.7891, el que indica que la intel·ligència artificial treballa bé un 78.91% dels casos. Si s'hagués comptat amb més temps, s'hauria millorat el model, tant augmentant el conjunt de dades d'entrada com provant encara més estructures de la xarxa neuronal o fins i tot augmentant el nombre d'epochs de l'entrenament, per tal d'aconseguir una exactitud més gran. No obstant això, tot i que el model no sempre és capaç de predir correctament les imatges, el resultat del treball és satisfactori, sobretot en pensar que tot això s'ha aconseguit partint de tan poc coneixement en l'àmbit.

[1] Random Access Memory.

Agraïments

En primer lloc vull agrair a la meva tutora, l'Antònia Subirà i el meu segon tutor en Joan Miarons, per confiar en mi des de l'inici i per aconsellar-me i guiar-me durant tot el procés que ha durat el treball.

També voldria agrair a l'Alba Blasco per ajudar-me a solucionar els problemes relacionats amb el text escrit i LaTeX i al meu germà, l'Arnau Llopart, per resoldre'm tots els dubtes relacionats amb conceptes de programació.

Finalment, m'agradaria agrair a La Fundació La Pedrera per brindar-me l'oportunitat de poder participar en Bojos per la Supercomputació i a l'Alberto Gutiérrez Torre per orientar-me al principi del treball.

Recursos electrònics

Nicholas Renotte (2021). *Tensorflow Object Detection in 5 Hours with Python* [En línia]. Recuperat 27 d'abril de 2021, des de:
`https://www.youtube.com/watch?v=yqkISICHH-U\&list=WL\&index=1\&t=11226s\&ab_channel=NicholasRenotte`

Udacity. *Intro to TensorFlow for Deep Learning* [En línia]. Recuperat 2 de juny de 2021, des de:
`https://classroom.udacity.com/courses/ud187`

TensorFlow (2019). *Get started with using TensorFlow to solve for regression problems (Coding TensorFlow)* [En línia]. Recuperat 5 de juny de 2021, des de:
`https://www.youtube.com/watch?v=-vHQub0NXI4\&ab_channel=TensorFlow`

Carlos Julio Pardo. *Overfitting y Underfitting Machine learning* [En línia]. Recuperat 7 de juny de 2021, des de:
`IntroducciÃşnaMachineLearningconTensorFlow`

Udacity. *AI Fundamentals* [En línia]. Recuperat 10 de juny de 2021, des de:
`https://classroom.udacity.com/courses/ud187`

AMP Tech (2017). *¿Cómo funcionan las redes neuronales?* [En línia]. Recuperat 12 de juny de 2021, des de:
`https://www.youtube.com/watch?v=IQMoglp-fBk\&t=247s\&ab_channel=AMPTech`

Dot CSV (2018). *¿Qué es una Red Neuronal? Parte 3 : Backpropagation* [En línia]. Recuperat 17 de juny de 2021, des de:
`https://www.youtube.com/watch?v=eNIqz_noix8\&ab_channel=DotCSV`

Josep Roca (2020). *¿Qué significan Entrenamiento e Inferencia en la Inteligencia Artificial?* [En línia]. Recuperat 18 de juny de 2021, des de:
`https://hardzone.es/tutoriales/reparacion/liquido-cafe-agua-dearramado-portatil/`

Miguel Ángel Hernández Castro (2018). *Introducción a Machine Learning con TensorFlow* [En línia]. Recuperat 18 de juny de 2021, des de:

https://www.adictosaltrabajo.com/2018/04/18/introduccion-a-machine-learning-con-tensorflow/

Udacity. *Introduction to Computer Vision* [En línia]. Recuperat 25 de juny de 2021, des de:
https://classroom.udacity.com/courses/ud810

Dot CSV (2020). *¡Redes Neuronales CONVOLUCIONALES! ¿Cómo funcionan?* [En línia]. Recuperat 1 de juliol de 2021, des de:
https://www.youtube.com/watch?v=V8j1oENVz00\&t=1s\&ab_channel=DotCSV

Ankit Paliwal (2018). *Understanding your Convolution network with Visualizations* [En línia]. Recuperat 3 de juliol de 2021, des de:
https://towardsdatascience.com/understanding-your-convolution-network-with-visualizations-a4883441533b

Google Developers. *ML Practicum: Image Classification* [En línia]. Recuperat 6 de juliol de 2021, des de:
https://developers.google.com/machine-learning/practica/image-classification/convolutional-neural-networks?hl=es\&authuser=1

Juan Ignacio Barrios Arce.*Redes Neuronales Convolucionales* [En línia]. Recuperat 15 de juliol de 2021, des de:
https://www.juanbarrios.com/redes-neurales-convolucionales/#Comparativa_entre_una_red_neuronal_tradicional_y_una_red_neural_convolucional

Redes Neuronales [En línia]. Recuperat 20 de juliol de 2021, des de:
https://ml4a.github.io/ml4a/es/neural_networks/

Diego Calvo (2017). *Red Neuronal Convolucional CNN* [En línia]. Recuperat 26 de juliol de 2021, des de:
https://www.diegocalvo.es/red-neuronal-convolucional/

Redes Neuronales [En línia]. Recuperat 20 de juliol de 2021, des de:
https://ml4a.github.io/ml4a/es/neural_networks/

Diego Calvo (2017). *Red Neuronal Convolucional CNN* [En línia]. Recuperat 26 de juliol de 2021, des de:
https://ichi.pro/es/funciones-de-activacion-99956793768697

Reinel (2018). *Mini ASL Alphabet* [En línia]. Recuperat 6 d'agost de 2021, des de:
https://www.kaggle.com/lcastrillon/mini-asl-alphabet

Keith Galli (2020). *Real-World Python Neural Nets Tutorial (Image Classification w/*

CNN) | Tensorflow & Keras [En línia]. Recuperat 6 d'agost de 2021, des de: https://www.youtube.com/watch?v=44U8jJxaNp8\&list=LL\&index=18\&t=1773s\&ab_channel=KeithGalli

Cudoo (2021). *American Sign Language alphabet for ASL beginners* [En línia]. Recuperat 9 d'agost de 2021, des de: https://cudoo.com/blog/american-sign-language-alphabet-for-beginners-in-asl/

Sakshi_Tiwari (2020). *Activation functions in Neural Networks* [En línia]. Recuperat 15 d'agost de 2021, des de: https://www.geeksforgeeks.org/activation-functions-neural-networks/i

Álvaro Artola Moreno (2019). *Activation functions in Neural Networks* [En línia]. Recuperat 20 d'agost de 2021, des de: http://bibing.us.es/proyectos/abreproy/92402/fichero/TFG-2402-ARTOLA.pdf

Rolandomr (2019). *American Sign Language Recogntion (ASL)* [En línia]. Recuperat 27 d'agost de 2021, des de: https://www.kaggle.com/rolandomr/american-sign-language-recogntion-asl

Codebasics (2020). *Activation Functions | Deep Learning Tutorial 8 (Tensorflow Tutorial, Keras & Python)* [En línia]. Recuperat 30 d'agost de 2021, des de: https://www.youtube.com/watch?v=icZItWxw7AI\&list=LL\&index=7\&t=162s\&ab_channel=codebasics

TensorFlow. *Overfit y underfit* [En línia]. Recuperat 2 de setembre de 2021, des de: https://www.tensorflow.org/tutorials/keras/overfit_and_underfit?hl=es-419

National Geographic (2021). *Breve historia visual de la inteligencia artificial* [En línia]. Recuperat 4 de setembre de 2021, des de: https://www.nationalgeographic.com.es/ciencia/breve-historia-visual-inteligencia-artificial_14419/15#slide-14#slide-14

Ringa Tech (2021). *Usa tus modelos de Tensorflow en páginas web | Exportación a Tensorflow.js* [En línia]. Recuperat 5 de setembre de 2021, des de: https://www.youtube.com/watch?v=JpE4bYyRADI\&list=LL\&index=13\&t=485s\&ab_channel=RingaTech

Juan Ignacio Barrios Arce (2019). *La matriz de confusión y sus métricas* [En línia]. Recuperat 14 de setembre de 2021, des de: https://www.juanbarrios.com/la-matriz-de-confusion-y-sus-metricas/

Dan Rasband (2018). *ASL Alphabet Test* [En línia]. Recuperat 25 de setembre de 2021,

des de:
`https://www.kaggle.com/danrasband/asl-alphabet-test`

Belal Elwikel (2019). *ASL_and_same_words* [En línia]. Recuperat 25 de setembre de 2021, des de:
`https://www.kaggle.com/belalelwikel/asl-and-some-words`

Ayush Thakur (2019). *ASL_and_same_words* [En línia]. Recuperat 25 de setembre de 2021, des de:
`https://www.kaggle.com/ayuraj/american-sign-language-dataset`

Omar Zakaria (2018). *ASL_alphabet* [En línia]. Recuperat 26 de setembre de 2021, des de:
`https://www.kaggle.com/omarzakariasalah/asl-alphabet`

Akash (2017). *ASL Alphabet* [En línia]. Recuperat 30 de setembre de 2021, des de:
`https://www.kaggle.com/grassknoted/asl-alphabet`

Generalitat de Catalunya (2019). *La intel·ligència artificial a Catalunya* [En línia]. Recuperat 11 d'octubre de 2021, des de:
`https://www.accio.gencat.cat/web/.content/bancconeixement/documents/informes_sectorials/informe-tecnologic-inteligencia-artificial.pdf`

IONOS (2020). *Unsupervised learning: aprendizaje automático sin restricciones* [En línia]. Recuperat 12 d'octubre de 2021, des de:
`https://www.ionos.es/digitalguide/online-marketing/marketing-para-motores-de-busqueda/unsupervised-learning/`

Sumit Saha (2018).*A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way* [En línia]. Recuperat 14 d'octubre de 2021, des de:
`https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53`

TensorFlow. *Introducción a los tensores* [En línia]. Recuperat 14 d'octubre de 2021, des de:
`https://www.tensorflow.org/guide/tensor`

GitHub (2020).*The 2020 State of the OCTO—VERSE* [En línia]. Recuperat 14 d'octubre de 2021, des de:
`https://octoverse.github.com/`

DeepLearningAI (2017). *Why Non-linear Activation Functions* [En línia]. Recuperat 16 d'octubre de 2021, des de:
`https://www.youtube.com/watch?v=NkOv_k7r6no\&list=LL\&index=6\&ab_channel`

=DeepLearningAI

Naren Castellon (2021). *Estructura de datos del tensor en Python* [En línia]. Recuperat 16 d'octubre de 2021, des de:
https://www.youtube.com/watch?v=6vOPjjHrWso\&list=LL\&index=5\&t=525s\&ab_channel=NarenCastellon

Annexos

Annex 1: Passos seguits

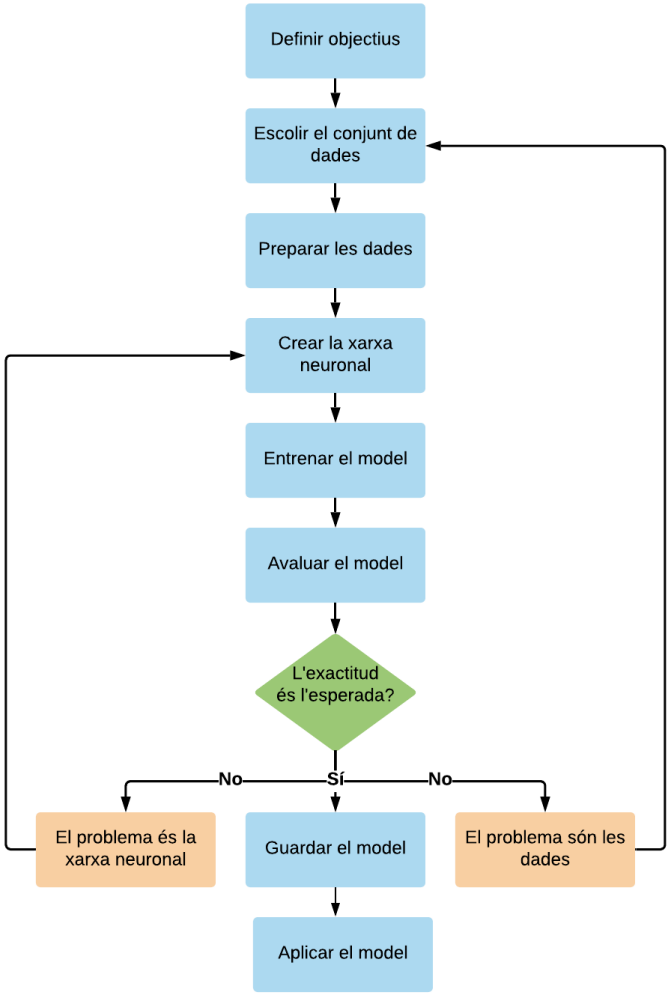


Figura 7.1: Passos seguits

Annex 2: Abecedari ASL

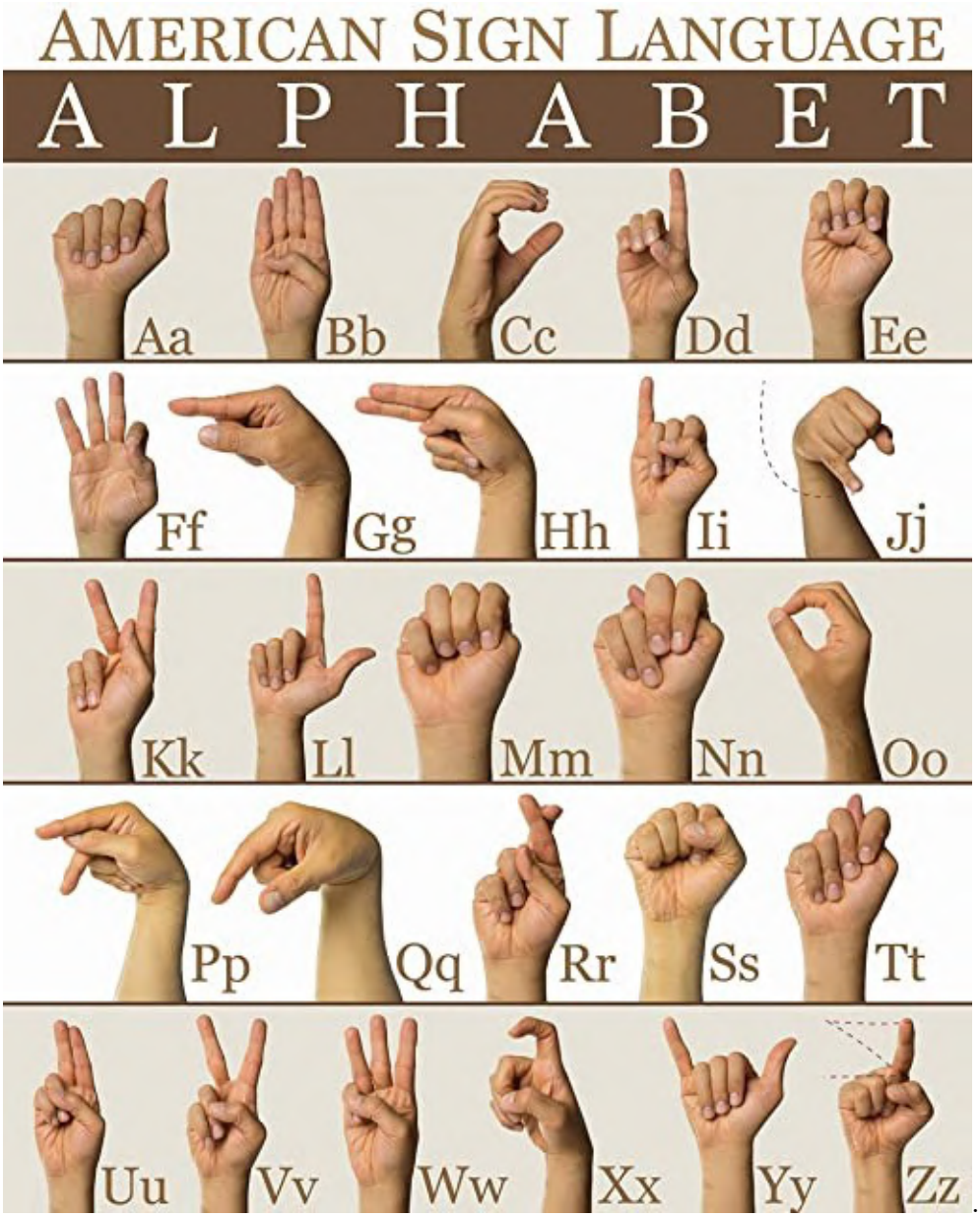


Figura 7.2: Abecedari ASL

Annex 3: Preparar les dades

```
import os
import cv2
from PIL import Image
import numpy as np
train_images=[]
labels=[]
a=0
train_path= os.listdir("/content/drive/MyDrive/entrenament")
for cdir in train_path:
  img=os.listdir("/content/drive/MyDrive/entrenament/"+cdir)
  for imatge in img:
    url="/content/drive/MyDrive/entrenament/"+cdir+"/" +imatge
    ig= cv2.imread(url)
    ig=cv2.resize(ig,(64,64), interpolation=cv2.INTER_CUBIC)
    arr= Image.fromarray(ig, 'RGB')
    labels.append(cdir)
    train_images.append(np.array(arr))


from sklearn.preprocessing import LabelEncoder
import pandas as pd
lb_encod= LabelEncoder()
labels= pd.DataFrame(labels)
labels= lb_encod.fit_transform(labels[0])

train_images= np.array(train_images)
np.save('imatge',train_images)
np.save('labels', labels)

train_images=np.load('imatge.npy')
labels=np.load('labels.npy')

import tensorflow as tf
train_images = tf.keras.utils.normalize(train_images, axis=1)



test_images=[]
test_labels=[]
a=0
test_path= os.listdir("/content/drive/MyDrive/examen")
for cdir in train_path:
img=os.listdir("/content/drive/MyDrive/examen/"+cdir)
for imatge in img:
url="/content/drive/MyDrive/examen/"+cdir+"/" +imatge
ig= cv2.imread(url)
ig=cv2.resize(ig,(64,64), interpolation=cv2.INTER_CUBIC)
arr= Image.fromarray(ig, 'RGB')
test_labels.append(cdir)
test_images.append(np.array(arr))
```

```
lb_encod= LabelEncoder()
test_labels= pd.DataFrame(test_labels)
test_labels= lb_encod.fit_transform(test_labels[0])

test_images= np.array(test_images)
np.save('test_imatge',test_images)
np.save('test_labels', test_labels)

test_images=np.load('test_imatge.npy', allow_pickle=True)
test_labels=np.load('test_labels.npy', allow_pickle= True)

test_images = tf.keras.utils.normalize(test_images, axis=1)
```

Annex 4: Xarxa neuronal

```
from keras import layers
from keras.layers import Dense, Conv2D, MaxPooling2D, Flatten,
                                    MaxPool2D, Activation

model= tf.keras.Sequential([
tf.keras.layers.Conv2D(64,(3,3), activation=tf.nn.relu, input_shape=(
                                    64,64,3)),
tf.keras.layers.MaxPooling2D((2,2), strides=2),

tf.keras.layers.Conv2D(64,(3,3), activation=tf.nn.relu),
tf.keras.layers.MaxPooling2D((2,2), strides=2),

tf.keras.layers.Flatten(),
tf.keras.layers.Dense(64,activation=tf.nn.relu),
tf.keras.layers.Dense(26,activation=tf.nn.softmax)
])

model.compile(optimizer='adam',
loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
metrics=['accuracy'])
```

Annex 5: Entrenar el model

```
history= model.fit(
train_images, labels,
batch_size= 32,
epochs= 100)
```

Annex 6: Avaluar i guardar el model

```
model.evaluate(test_images, test_labels)


model.save('MFX.h5')
model.save_weights('MFWX.h5')
```

Annex 7: Aplicar el model

```
import tkinter as tk
from tkinter import filedialog
from PIL import Image
from PIL import ImageTk
import cv2
import numpy as np
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
from keras_preprocessing.image import img_to_array

image=None

def escollir_imatge():
#especificar el tipus d'arxiu que es pot seleccionar
  path_image = filedialog.askopenfilename(filetypes=[
  ('image', '.jpg'),
  ('image','.jpeg'),
  ('image','.png')])


  if len(path_image) >0:
    img= Image.open(path_image)
    image= img.resize((250,210))
    render= ImageTk.PhotoImage(image)
    img1= tk.Label(root, image=render)
    img1.image=render
    img1.place(x=74, y=80)

    #preparar imatge:
    img=plt.imread(path_image)
    img=cv2.resize(img,(64,64), interpolation=cv2.INTER_CUBIC)
    img= img_to_array(img)
    img=np.expand_dims(img, axis=0)
    img_prep= tf.keras.utils.normalize(img, axis=1)

    #importar model
    model= keras.models.load_model('MFX.h5')
    model.load_weights('MFWX.h5')
```

```
    #predir la lletra de la imatge
    predictions = model.predict(img_prep)
    predicted_value= np.argmax(predictions)
    new_predicted_value=predicted_value.tolist()
    print(new_predicted_value)

    lletres={0:"A",1:"B",2:"C",3:"D",4:"E",5:"F",
        6:"G",7:"H",8:"I",9:"J", 10:"K",
        11:"L",12:"M",13:"N",14:"O",15:"P",
        16:"Q",17:"R",18:"S",19:"T",20:"U",
        21:"V",22:"W",23:"X",24:"Y",25:"Z"}

    lletra= lletres[new_predicted_value]

    labelI=tk.Label(root, text="Pertany a la lletra:")
    labelI.place(x=120,y=305,width=160,height=35)
    labelI.config(font=("Verdana",12))

    solucio=tk.Label(root, text=lletra)
    solucio.place(x=135,y=345,width=130,height=35)
    solucio.config(font=("Verdana",12))


root=tk.Tk()
root.title('Abecedari ASL')
root.geometry('400x500')

lblInputImage = tk.Label(root)
lblInputImage.place(x=155,y=220,width=50,height=35)


btnImatge= tk.Button(root, text='Escollir imatge' , command=
                                    escollir_imatge)
btnImatge.config(fg="white", bg="blue",font=("Arial",10))
btnImatge.place(x=116,y=400,width=170,height=45)


lbl=tk.Label(root, text="Abecedari ASL")
lbl.place(x=115,y=32,width=170,height=50)
lbl.config(font=("Book Antiqua",18))

root.mainloop()
```

Podeu trobar els còdis anteriors al meu perfil de GitHub a través d'aquest enllaç: `https://github.com/nuriallfe/Intel-lig-ncia-artificial.git`.