Deep Learning per la conducció autònoma de vehicles

Document:
Memòria

Autor:
Carles Montilla Garcia

Director /Co-director:
Jose Antonio Soria Perez

Titulació:
Grau en Enginyeria en Vehicles Aeroespacials

Convocatòria:
Primavera

TREBALL FINAL D'ESTUDIS

Treball de Fi de Grau

Deep Learning per la conducció autònoma de vehicles

Carles Montilla Garcia

22 de Juny de 2021

Universitat Politècnica de Catalunya
Enginyeria en Vehicles Aeroespacials

Director
Jose Antonio Soria Perez

Data de lliurament
22 de Juny de 2021

Documents

1. **Memòria**
2. Pressupost

Resum

Cada any aproximadament 1,35 milions de persones moren per accidents de trànsit, el 94% d'aquests com a resultat d'errors humans. El desenvolupament de cotxes autònoms és un dels àmbits més populars i de moda on s'està experimentant amb tècniques de Deep Learning per tal de garantir la seguretat i la integritat de les persones. El Deep Learning és un subconjunt de tècniques en l'àmbit del *Machine Learning* que serveix per entrenar màquines algorítmiques a realitzar tasques sense haver sigut programades de manera explícita.

L'objectiu d'aquest treball consisteix en preparar un vehicle de radio-control i programar la seva unitat computacional perquè, mitjançant la utilització de tècniques d'aprenentatge profund supervisat, aprengui a moure's en un circuit tancat. Per tal d'assolir-lo es realitzaran diferents experiments en els quals es modificaran els paràmetres del vehicle així com les dades emprades en l'entrenament. En cada experiment es realitzarà una conducció diferent: sentit únic, ambdós sentits o ambdós sentits amb entrada de pista.

A partir dels resultats dels experiments s'obtenen els errors del model en funció de les èpoques d'entrenament. Es pot observar que, primer de tot, perquè un model sigui vàlid, a més a més de tenir un error baix a l'etapa d'entrenament l'ha de tenir a l'etapa de validació. En el cas de que l'error en l'etapa de validació sigui major, s'obté un model massa complex (model sobreajustat) i, per tant, no vàlid. Finalment, s'arriba a la conclusió de que el correcte entrenament d'un model depèn majoritàriament de la qualitat de les dades d'entrada, així com de la quantitat d'aquestes, i no del nombre d'èpoques. La qualitat de les dades dependrà principalment de la conducció i la preparació, sent aquesta última un filtratge i etiquetatge de les sortides.

Abstract

Approximately 1.35 millions of people die in traffic accidents each year, 94% of them as a result of human error. The development of autonomous cars is one of the most popular and trendy areas where Deep Learning techniques are being experimented with in order to ensure the safety and integrity of people. Deep Learning is a subset of techniques in the field of Machine Learning that is used to train algorithmic machines to perform tasks that have not been explicitly programmed.

The aim of this project is to prepare a radio-controlled vehicle and program its computing unit so that, through the use of supervised Deep Learning techniques, it can learn to move in a closed circuit. In order to achieve so, different experiments will be carried out in which the parameters of the vehicle will be modified as well as the data used in the training. In each experiment a different type of driving will be performed: one way, both directions or both directions with track entry.

From the results of the experiments, the model errors are obtained according to the training epochs. First of all, it can be observed that for a model to be valid, in addition to having a low error in the training stage, it must have it in the validation stage. In the event that the error in the validation stage is greater, it is obtained a model that is too complex (overfitting model) and therefore invalid. Finally, it is concluded that the correct training of a model depends largely on the quality of the input data, as well as its quantity, and not the number of epochs. The quality of the data will depend mainly on the driving and the preparation, the latter being a filtering and labelling of the outputs.

Índex

Índex de figures

Índex de taules

Nomenclature

CNN	Convolutional Neural Networks
CNN	Convolutional Neural Networks
DL	Deep Learning
GPUs	Graphic Processing Units
IA	Intel·ligència Artificial
ML	Machine Learning
PWM	Pulse Width Modulation
ReLu	Rectifier Linear Unit
RNN	Recurrent Neural Networks
RNN	Recurrent Neural Networks
RPTT	Retropropagació a través del temps
SD	Secure Digital
SSH	Secure Shell
USB	Universal Serial Bus
WiFi	Wireless Fidelity

1 Introducció

1.1 Antecedents

El Deep Learning és un subconjunt de tècniques en l'àmbit del *Machine Learning* (ML) que serveix per entrenar màquines algorítmiques a realitzar tasques sense haver sigut programades de manera explícita [1]. El desenvolupament de cotxes autònoms és un dels àmbits més populars i de moda on s'està experimentant amb l'aplicació de tècniques d'Intel·ligència Artificial (IA) [2]. Així al 2020, empreses com Waymo o AutoX d'Alibaba han creat serveis de taxis amb flotes de cotxes autònoms sense conductor de seguretat. Amb l'avenç en les tècniques d'aprenentatge profund s'estan substituint ràpidament a les persones com a conductors [3].

Aquests vehicles identifiquen objectes, interpreten situacions, i prenen decisions basades en la detecció d'objectes i algorismes *software* de classificació. Ho fan perquè disposen de diferents tipus de sensors, com càmeras, radar and lidar que, combinat amb els algorismes de IA actuen com els ulls i el cervell humà per interpretar senyals a la carretera, identificar línies o reconèixer encreuaments. Així, aquest àmbit ha tingut un creixement important gràcies a que s'ha beneficiat d'aquestes tècniques que s'implementen sobre sistemes encastats [2].

El principal objectiu de la conducció autònoma és la seguretat i la integritat de les persones, ja que s'estima que el 94% dels accidents són com a resultat d'errors humans, arribant a 1,35 milions de morts per accidents de trànsit a tot el món i 20 milions de lesionats, i la conducció autònoma pot ajudar a reduir aquests accidents [4].

Aquesta seguretat pot incrementar encara més si els vehicles autònoms cooperen entre ells per gestionar millor el trànsit. Això pot contribuir en determinades sistuacions a descongestionar el trànsit, reduint embossos que dificulten la circulació dels vehicles, i pal·liar el canvi climatic, reduint la contaminació, al mateix temps que s'agilitza el transport de mercaderies i persones. A més, l'ús de bateries contribueix a reduir la contaminació per gasos d'efecte hivernacle, com el CO_2 [3].

L'acceptació d'aquestes tecnologies comporta un canvi de paradigma sense precedents, tant a nivell filosòfic i ètic com econòmic. D'una banda, molts països no tenen les seves vies adaptades per a cotxes autònoms i no autònoms particulars i, adaptar-les, requereix de grans inversions per part dels governs. D'altra banda, la conducció autònoma suposa una linea més de negoci per empreses però, al mateix temps, condiciona laboralment el lloc de treball en el sector de transport de mercaderies i persones [3].

Per capacitar als cotxes en la presa de decisions per la conducció autònoma, els algorismes desenvolupen models matemàtics que s'entrenen a partir de conjunts de dades obtingudes de la vida real. En aquest sentit, el ML comprèn dos grups de tècniques principals: l'aprenentatge supervisat i no supervisat. Al model no supervisat, l'algorisme rep dades sense classificar i sense instruccions de com processar-les, de manera que ha d'esbrinar què ha de fer tot sol. En canvi, a l'aprenentatge supervisat, al model se li donen instruccions de com ha d'interpretar les dades. Aquest és l'escenari més utilitzat en cotxes autònoms, ja que permet entrenar a l'algorisme amb dades classificades [2].

1.2 Objectius

L'objectiu d'aquest treball consisteix en preparar un vehicle de radio-control (RC) i programar la seva unitat computacional perquè, mitjançant la utilització de tècniques d'aprenentatge profund supervisat, aprengui a moure's en un circuit tancat.

1.3 Abast

Es disposa de tots els accessoris d'un vehicle RC, tipus Donkey Car (Figura 1.1), amb una unitat computacional esquestada, i un circuit tancat en lona de plàstic amb línies vials. El vehicle RC disposa d'una camèra que s'ha acoblat i queda fixada al xasis de manera que pot capturar les marques de terra mitjançant instantànies fotogràfiques.

La unitat permet accionar la direcció de les rodes davanteres així com els motors que permeten moure el vehicle per avançar i retrocedir. El treball engloba l'acoblament del vehicle així com la seva posada en marxa, i la utilització de scripts de *Machine Learning* perquè el vehicle aprengui a guiar-se amb la càmera pel camí traçat en dos escenaris: circulació en tots dos sentits de la marxa i ingrés al circuit traçat.

Figura 1.1: Vehicle RC Donkey-Car. Font: [5]

Per aconseguir aquests objectius, s'apliquen tècniques d'aprenentatge supervisat que inclouen les fases següents:

1. **Obtenció de les dades**

   En aquesta fase s'obtenen les dades corresponents que els algorismes de IA han de processar i que, a priori, serveixen per resoldre el problema a tractar. Per aquesta aplicació, es necessita operar el vehicle RC manualment, de manera que vagi obtenint fotografies del circuit i, al mateix temps, capturi els valors del servos de la direcció i l'accionament dels motors DC. Les fotografies s'utilitzen com entrada del model IA (fase 3), mentre que servos i motor DC corresponen al *target*, o bé sortida del model.

2. **Preparació**

   Obtingudes les dades, el següent pas consisteix en preparar-les perquè l'algorisme escollit a la fase 3 les processi correctament. Operacions típiques en aquesta fase inclouen: el filtratge de les imatges, etiquetatge de les sortides i augmentació (que pot ser opcional)

El filtratge és una operació molt comuna que consisteix en descartar imatges amb característiques del conjunt de dades que estadísticament puguin tenir valors atípics i que poden condicionar el model en la fase d'aprenentatge i limitar la seva precisió a posteriori.

A diferència de l'aprenentatge no supervisat, on l'objectiu és trobar distribucions o classes d'entrada, l'aprenentatge supervisat requereix etiquetar cada mostra amb la classe corresponent perquè el model pugui detectar-la. Aquí, l'estratègia consisteix en utilitzar caràcters numèrics per representar cada classe (classe 1, classe 2,... classe n), o paraules representatives. En aplicacions d'imatge, l'etiquetatge sovint requereix de l'ús de programari alternatiu per marcar el contorn d'objectes que apareixen en les instantànies mitjançant màscares.

Una altra operació que sovint es realitza en la preparació de dades quan no es disposen de moltes dades és l'augmentació (també conegut com a *bootstrap*). Aquesta operació consisteix en incrementar el nombre de mostres del conjunt inicial de manera artifical, però modificant algun paràmetre o característica d'entrada. En aplicacions de detecció d'objectes en imatges, per exemple, generar imatges amb l'angle de referència modificat ajuda als models classificadors a generalitzar el problema i millorar la classificació.

En l'aplicació que ocupa a aquest treball, per obtenir suficients mostres s'han realitzat varis intents de conducció manual, mitjaçant comandament remot, i s'han ajuntat totes les imatges de cada intent en un únic directori. En quan a la informació sobre el *target*, aquesta s'escriu en arxius independents adaptant el codi que s'executa durant la prova de conducció manual.

Aquesta és una fase que requereix de treball exhaustiu que, de vegades, suposa una despesa econòmica i temps considerables segons l'aplicació desenvolupada.

3. **Conjunts d'entrenament i avaluació**

   Amb les imatges i les etiquetes del *target* en arxius independents, es separen en dos grups: Entrenament i Avaluació. El primer conjunt serveix per configurar i adaptar el model perquè realitzi la tasca de classificació desitjada. D'aquí el nom de «entrenament», doncs necessita tant de les dades com les etiquetes amb les classes per tenir una referència de la desviació i actualitzar els seus paràmetres per corregir-los. En el conjunt d'avaluació, les etiquetes s'utilitzen per determinar si la predicció del model, que en aquest cas correspon als accionaments del vehicle, és correcta. Un criteri habitual és el 80-20, que bàsicament consisteix en agafar el 80% de les dades inicials com a conjunt d'entrenament, i el 20% restant d'avaluació.

4. **Configuració del model**

   Amb els conjunts d'entrenament, el següent pas és escollir el model (o algorisme de IA) que millor s'adapta per resoldre el problema. Existeixen moltes tècniques de classificació. Mètodes populars que funcionen amb dades numèriques i cadenes de paraules són: *k-Nearest Neighbors*, CART (*Classification and Regression Trees*), LVQ (*Learning Vector Quantization*), NN (*Neural Networks*) o SVM *(Support Vector Machines*), entre d'altres. Entre els algortimes que s'utilitzen per imatge, darrerament s'han fet molt populars les xarxes R-CNN (*Recursive-Convolutional Neural Networks*).

5. **Entrenament**

   En aquesta fase, el model utilitza les dades d'entrenament com a entrada per calcular la seva predicció (o valors de sortida). Mitjançant les etiquetes, es pot determinar la desviació en la predicció de les accions de control del vehicle; mitjançant algorismes d'optimització, determinar els valors ens els paràmetres del model que generen les accions correctes.

6. **Predicció de les accions**

   Amb els paràmetres del model configurats després de l'entrenament, el següent pas consisteix en utilitzar el conjunt d'avaluació en el model per calcular les accions que es requereixen en el vehicle perquè circuli correctament pel circuit. Aquí, l'objectiu de les etiquetes no és el

de modificar el model (com en el pas anterior), sinó comparar-les amb l'etiquetatje realitzat al punt 2 i veure si el model respon correctament. Per aquest motiu, és important que les imatges introduïdes a l'entrada del model en aquesta fase siguin diferents i no s'hagin usat durant l'entrenament.

7. **Avaluació i millora**

   Si a nivell estadístic el comportament del model no és el desitjat, el següent pas natural és revisar algun dels punts que intervenen en la configuració de les fases anteriors i tornar a repetir l'experiment.

1.4 Requeriments

Sota aquestes premises, aquest projecte planteja l'ús d'eines amb les següents prestacions:

1. **DonkeyCar i circuit de proves**

   Es disposa d'un vehicle RC tipus DonkeyCar de dimensions 20 cm d'amplada per 25 cm de llargada i 15 cm d'alçada. Inclou un servomotor MG996R amb un angle de gir de 120 graus (60 en cada direcció) i un parell de motor de 9 kg/cm, un motor 37-520 DC amb una relació de reducció 1:10 i un ralentí de 740 rpm, i una càmera de 5 MP de resolució i 160 graus de camp de visió. El vehicle adquireix una velocitat màxima de 3,5 m/s. La plataforma utilitzada és una Raspberry Pi 4 Model B de 4 GB de RAM amb un processador Broadcom BCM2711 de 1,5 GHz i un total de 40 pins GPIO (General Purpose Input/Output). La font d'alimentació és de 12,6 V, amb tres bateries 18650 connectades en sèrie que proporcionen un temps de funcionament d'entre 5 i 20 minuts.

   El circuit de proves és una lona d'unes dimensions de 2 metres d'amplada per 3 metres de llargada que inclou un traçat en forma de 0 de 5,65 metres de recorregut.

2. **Bases de dades d'imatges i etiquetatge**

   Per obtenir les imatges, s'ha operat el vehicle de manera manual sobre el circuit de proves. S'ha adaptat el software per adquirir instantànies del recorregut a una cadència de 18 fotografies/segon a color amb una resolució de 5 MP i escrivint en arxius JSON (*Java Script Object Notation*) les sortides, o *targets* del vehicle que permeten operar-lo: els valors dels senyals del PWM dels servos de direcció i els motors DC. Les imatges són en format JPEG (*Joint Photographic Expert Group*) i tenen una resolució de pixels de 160 x 120. Donat que la capacitat de la bateria no permet operar el vehicle el temps com per adquirir suficients imatges, en cada experiment s'han realitzat diverses conduccions fins arribar a obtenir un total d'entre 8000 i 10000 imatges que s'han utilitzat per entrenar el model.

3. **Model de predicció**

   Per l'aprenentatge del vehicle, s'utilitza un model basat en CNN (*Convolutional Neural Network*) que permet operar amb imatges d'entrada i genera dues sortides amb valors reals que corresponen als senyals d'actuació del servomotor de direcció i dels motors DC de les rodes traseres. CNN és una xarxa neuronal profunda destinada a resoldre problemes de classificació i regressió d'aprenentatge automàtic amb imatges [6] i que, en l'aplicació d'aquest treball s'utilitza com a model de regressió. Per l'entrenament de la xarxa neuronal s'inicialitzen els pesos en base a la llibreria COCO per tal d'accelerar l'entrenament. Les llibreries per treballar amb aquest model s'han desenvolupat en llenguatge Python i estan disponibles a *Github* [7]. Per lo que es poden utilitzar en diferents plataformes Linux, Windows i MAC.

4. **Entrenament i Inferència**

   L'algorisme d'entrenament s'ha executat sobre ordinador portàtil amb sistema operatiu Windows i, una vegada obtingut el model, aquest s'ha carregat sobre la Raspberry Pi del DonkeyCar. L'ordinador incorpora una CPU i7 de INTEL de 1,8 GHz, amb 8 GBytes de RAM. S'ha instal·lat el programari Anaconda que inclou els paquets software, tant interpret de Python com l'editor Spider. No obstant, com que la Raspberry Pi opera amb sistema Linux Raspi OS, s'ha decidit instal·lar Vrtual Box per crear una màquina virtual i poder compilar els codis que van a la Raspberry Pi. L'entorn virtual generat, s'ha configurat amb una RAM de 5 GBytes per agilitzar l'entrenament de la CNN.

5. **Precisió del model**

   Per avaluar el comportament del model neuronal, s'han fet diferents experiments de conducció autònoma: circulació horaria, antihorari i entrada a pista. A més a més, s'ha desenvolupat un segon criteri d'avaluació on el model entrenat realitza prediccions a partir de noves dades d'entrada. El vehicle, segueix el traçat del circuit en els dos sentits de la marxa. S'ha realitzat un experiment provant l'entrada del vehicle al circuit per diferents localitzacions, on en un 25% dels intents ho ha realitzat amb éxit.

   Totes les eines software estan basades en llenguatge Python. Les llibreries utilitzades pel DonkeyCar i el funcionament del model de Deep Learning implementat han estat Tensorflow, Numpy, Coco, Pillow, Tornado, Scipy, Keras, Matplotlib, H5py, Scikit-image i OpenCV, entre d'altres.

1.5 Justificació i Motivació

Des de petit el món de la robòtica ha causat un gran interés en mi. Abans de començar a cursar el grau en Enginyeria de Vehicles Aeroespacials havia format part en alguns projectes relacionats amb aquesta temàtica, com per exemple la creació d'una mà robòtica o un cotxe controlat per veu. Tanmateix, al llarg d'aquests darrers anys només he cursat un parell d'assignatures on es tractaven temes relacionats amb l'electrònica.

No va ser fins el quatrimestre passat quan vaig tenir la possibilitat de realitzar optatives de robòtica i automatització, les quals van servir per adonar-me que volia ampliar els meus coneixements sobre aquesta indústria i, conseqüentment, realitzar un projecte de fi de grau relacionat amb la robòtica.

A causa de la manca d'assignatures de programació en el grau, els meus coneixements en aquest aspecte eren pràcticament nuls. El treball de fi de grau va ser una altra oportunitat per adquirir capacitats de programació i introduir-me al món del llenguatge Python.

Un cop decidit el camp del treball havia d'escollir la temàtica del mateix. Aquest últim any els vehicles autònoms han adquirit una gran popularitat, principalment per les noticies relacionades amb la marca de cotxes Tesla i el seu director executiu, Elon Musk. Conversant amb el meu germà em va informar sobre el Deep Learning i l'existència de cotxes autònoms d'escala reduïda, concretament del DonkeyCar. Després de reunir-me amb el meu tutor i comentar-li el meu interés pel DonkeyCar, vam decidir realitzar un treball sobre el Deep Learning per la conducció autònoma de vehicles.

Tot i ser un projecte molt simple en comparació amb els cotxes autònoms actuals, crec que és una molt bona oportunitat per adquirir els fonaments bàsics de la conducció autònoma i qui sap si en un futur poder treballar en alguns dels camps relacionats.

1.6 Entregables i continguts de la memòria

Aquest treball consta de dos entregables: la memòria i el document del pressupost del projecte. La memòria està formada per un total de nou capítols. Aquest capítol ha presentat les directrius generals del projecte, així com l'objectiu principal d'aquest. El capítol 2 introdueix els fonaments teòrics i conceptes principals del Machine Learning i, sobretot, el Deep Learning, que s'aplica a la conducció autònoma. També es mencionen eines hardware i software que s'utilitzen a l'actualitat pel desenvolupament de vehicles autònoms a petita escala com el DonkeyCar.

Al capítol 3 s'indiquen les decisions que s'han pres per aconseguir els objectius del treball. El capítol 4 desenvolupa el muntatge de l'estructura, instal·lació i configuració del software en detall per tal de poder posar en marxa el vehicle. D'altra banda, al capítol 5 es recull tota la part del treball relacionada amb l'obtenció de dades i l'entrenament dels models.

L'anàlisi dels resultats obtinguts en l'entrenament dels models es troba al capítol 6, on es realitza un estudi comparatiu entre els quatre models seleccionats. D'altra banda, al capítol 7 es mostra un resum del pressupost total del treball, incloent tant el preu del material com dels recursos humans necessaris pel desenvolupament d'aquest.

Finalment, a l'últim capítol (capítol 8), a més a més de les conclusions finals s'expliquen quins passos s'haurien de seguir per tal de millorar el projecte actual.

2 Antecedents i revisió de l'estat de l'art

2.1 Deep Learning

El Deep Learning (DL) és una branca del *Machine Learning* (ML) que, a la vegada, és una branca de la Intel·ligència Artificial (IA). Per tal de definir i entendre què és el Deep Learning, doncs, s'han de definir prèviament els conceptes de *Machine Learning* i Intel·ligència Artificial.

La Intel·ligència Artificial és una ciència enfocada a la creació d'algoritmes amb la finalitat d'informar de futures prediccions. Dit d'una altra manera, és una ciència que engloba totes les tècniques que permeten als ordinadors imitar comportaments humans. D'altra banda, el *Machine Learning* és una part de la Intel·ligència Artificial que es basa en ensenyar a un algoritme a aprendre a partir d'experiències sense haver sigut prèviament programat.

Finalment, el Deep Learning és la part del Machine Learning que es centra en utilitzar xarxes neuronals amb la finalitat d'obtenir models o patrons a partir de dades no processades (en anglés, *raw data*) i, posteriorment, utilitzar aquests models per aprendre a realitzar tasques.

Tot i que els fonaments del Deep Learning, juntament amb les xarxes neuronals, han existit des de fa dècades, no ha sigut fins la actualitat quan realment el Deep Learning ha adquirit importància. El principal motiu ha sigut la globalització o el domini de les dades. Les tècniques i els models emprats requereixen una enorme quantitat de dades i actualment ens trobem en una època en la qual s'ha facilitzat la recol·lecció i l'emmagatzematge de dades. D'altra banda, els algoritmes poden beneficiar-se del *hardware* actual (GPUs) i obtenir una eficàcia i eficiència major [1].

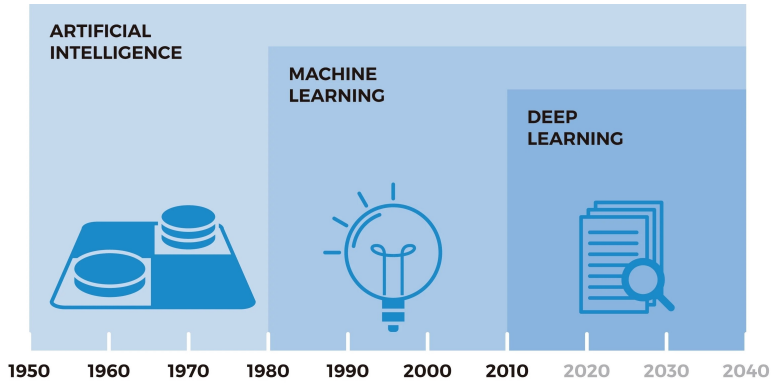


Figura 2.1: Evolució de la Intel·ligència Artificial. Font (modificada): [8]

2.2 Tipus d'aprenentatges

Abans d'endinsar-se en els algoritmes o les xarxes neuronals emprades en aquestes tècniques, és important entendre les tres categories de Machine Learning que existeixen. Aquestes categories són: aprenentatge supervisat, no supervisat i per reforç.

2.2.1 Aprenentatge supervisat

El tipus de Machine Learning més comú, tant si és Deep Learning com si no, és l'aprenentatge supervisat. Aquest aprenentatge es centra en el desenvolupament de patrons a partir de la relació entre variables i resultats coneguts i, a més a més, treballar amb bases de dades etiquetades.

Per aconseguir-ho, l'ordinador o la màquina és «alimentada» amb un mostratge de dades amb certes característiques, juntament amb el valor correcte del resultat. El fet de conèixer els valors dels resultats i de les característiques de les dades classifica la base de dades com a «etiquetada». Un cop la màquina ja ha obtingut la informació, l'algoritme desxifra patrons existents en les dades i crea un model que és capaç de reproduir les mateixes decisions o regles amb noves dades, predient d'aquesta manera resultats no coneguts prèviament [9].

L'aprenentatge supervisat s'utilitza principalment en dos tipus de problemes [10]:

- Classificació: l'algoritme s'encarrega de dividir les dades proporcionades en diferentes categories.
- Regressió: l'algoritme s'encarrega d'entendre la relació entre variables dependents i independents. Aquest tipus de models són útils a l'hora de predir valors numèrics a partir dels diferents valors de dades.

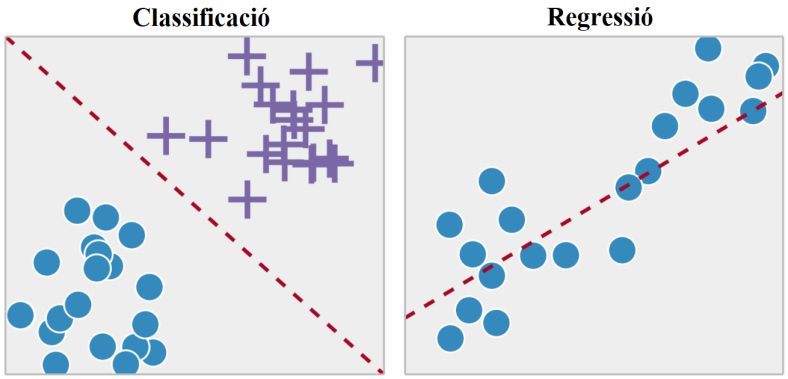


Figura 2.2: Tipus de models en l'aprenentatge supervisat. Font (modificada): [11]

2.2.2 Aprenentatge no supervisat

L'aprenentatge no supervisat utilitza algoritmes per analitzar i agrupar bases de dades no etiquetades (no es coneixen els valors dels resultats). A diferència de l'aprenentatge supervisat, aquests algoritmes descobreixen patrons ocults en les dades sense la supervisió o intervenció del ser humà (d'aquí el nom de «no supervisat»). Pel fet de no utilitzar bases de dades etiquetades, no hi ha una manera específica de comparar l'actuació del model en la majoria de mètodes dins l'aprenentatge no supervisat [10, 11, 12].

En aquest cas, els algoritmes emprats permeten realitzar tasques molt més complexes:

- Clusterització (en anglés, *clustering*): és una tècnica que permet classificar dades no etiquetades en grups basats en les seves similituds o diferències. És l'equivalent al mètode de classificació en l'aprenentatge supervisat.
- Associació: és un mètode que utilitza diferentes normes o decisions amb l'objectiu de trobar relacions entre les variables dins d'una base de dades.

- Reducció dimensional: és una tècnica emprada quan el número de característiques (o dimensions) en una base de dades és massa elevat i, per tant, complex. Amb aquesta tècnica s'aconsegueix reduir el número de dades d'entrada a una quantitat més manejable sense fer malbé la integritat de les dades.

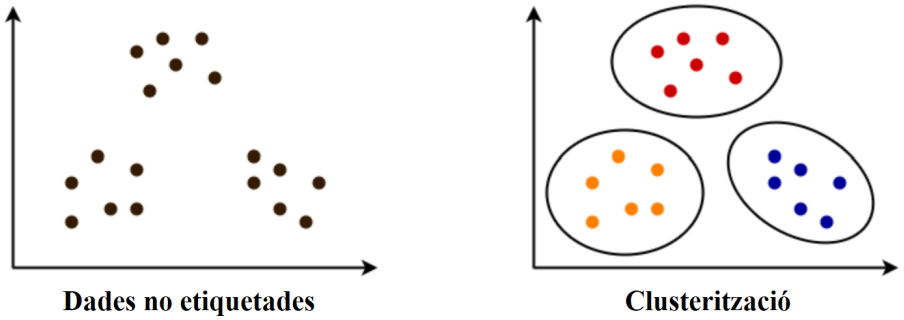


Figura 2.3: Clusterització en l'aprenentatge no supervisat. Font (modificada): [13]

2.2.3 Aprenentatge per reforç

L'aprenentatge per reforç és la categoria més avançada del Machine Learning. A diferència dels dos casos anteriors, aquest tipus d'aprenentatge millora constantment el seu model a partir de la informació obtinguda en iteracions anteriors [9].

A la Figura 2.4 es pot observar una representació bàsica, juntament amb els elements implicats, d'un model d'aprenentatge per reforç.

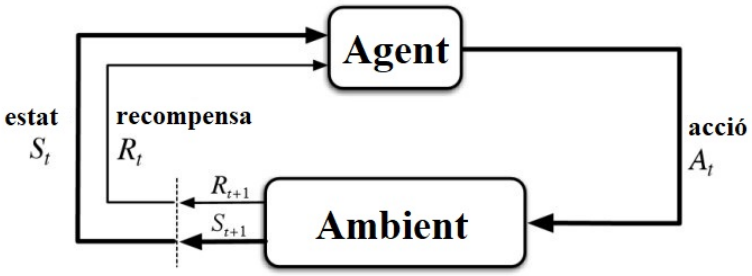


Figura 2.4: Esquema bàsic de l'aprenentatge per reforç. Font (modificada): [14]

Aquest aprenentatge té quatre elements essencials [14, 15]:

- Agent: el programa que entrena amb l'objectiu de realitzar la tasca especificada.
- Ambient o entorn: el món, ja sigui real o virtual, en el qual l'agent realitza les accions.
- Acció: és el moviment realitzat per l'agent que provoca un canvi d'estat en l'entorn.
- Recompenses: és l'avaluació d'una acció, que pot ser postiva o negativa. És la informació que es rep per part de l'ambient.

2.3 Xarxes neuronals

2.3.1 Perceptró

El perceptró és la xarxa neuronal més bàsica que existeix dins l'aprenentatge supervisat i, per tant, el pilar fonamental del Deep Learning. El seu origen va tenir lloc l'any 1958 pel psicòleg Frank

Rosenblatt i consistia en un classificador binari que era capaç de reconèixer patrons a partir d'un entrenament amb dades [16].

El funcionament d'un perceptró s'inicia amb els valors de les dades d'entrades que són multiplicats pels seus pesos corresponents. Després, aquests valors són sumats entre si, obtenint un resultat que és introduït a la funció d'activació i produint així el valor de la sortida. Aquest funcionament pot ser observat matemàticament a l'equació (2.1) o, d'altra banda, gràficament a la Figura 2.5. És important fixar-se que el pes d'una entrada indica la força o repercussió d'aquesta, així com la constant permet desplaçar la funció d'activació verticalment [17].

$$\hat{y} = g\left(w_0 + \sum_{i=1}^{n} x_i w_i\right) \tag{2.1}$$

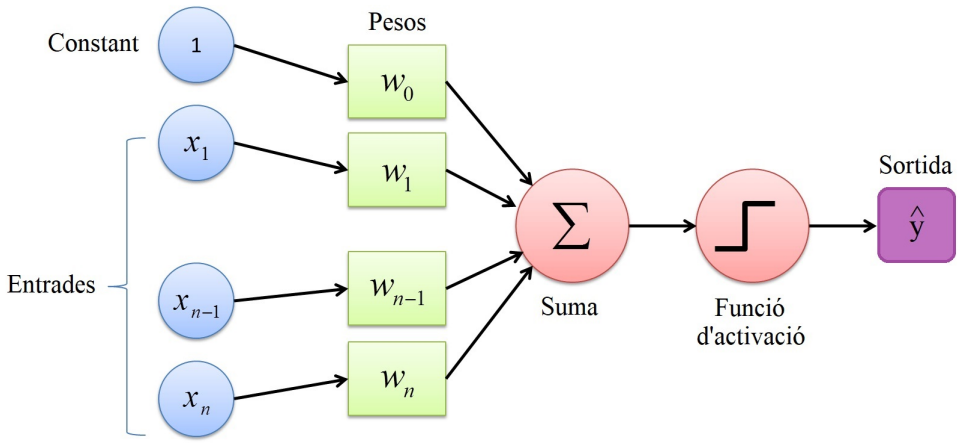


Figura 2.5: Representació gràfica d'un perceptró simple o mononeuronal. Font (modificada): [17]

2.3.1.1 Funció d'activació

La funció d'activació és una funció no-lineal que té la finalitat d'introduir no-linealitats a la xarxa neuronal i, per tant, poder tractar amb dades no-lineals. Aquest fet és molt important ja que pràcticament totes les dades existents no són lineals [1].

2.3.2 Perceptró multicapa

Un perceptró multicapa és aquell en el qual hi ha una o més capes ocultes entre les entrades i les sortides. S'anomena precisament «capa oculta» perquè, a diferència de les capes d'entrada i sortida, els estats d'aquesta no són visibles des de l'exterior del sistema. La capa oculta és la que s'encarrega d'aplicar els pesos corresponents a les dades d'entrada (equació (2.2)) i, a través d'una funció d'activació, enviar els resultats a les sortides (equació (2.3)) [1, 18].

La característica principal d'aquesta xarxa neuronal, com bé es pot observar a la Figura 2.6, és que cada neurona d'una capa (ja sigui d'entrada o oculta) està connectada amb cadascuna de les neurones de la capa següent.

$$z_i = w_{0,i}^{(1)} + \sum_{j=1}^{n} x_j w_{j,i}^{(1)} \tag{2.2}$$

$$y_i = g\left(w_{0,i}^{(2)} + \sum_{j=1}^{d_1} g(z_j) w_{j,i}^{(2)}\right) \tag{2.3}$$

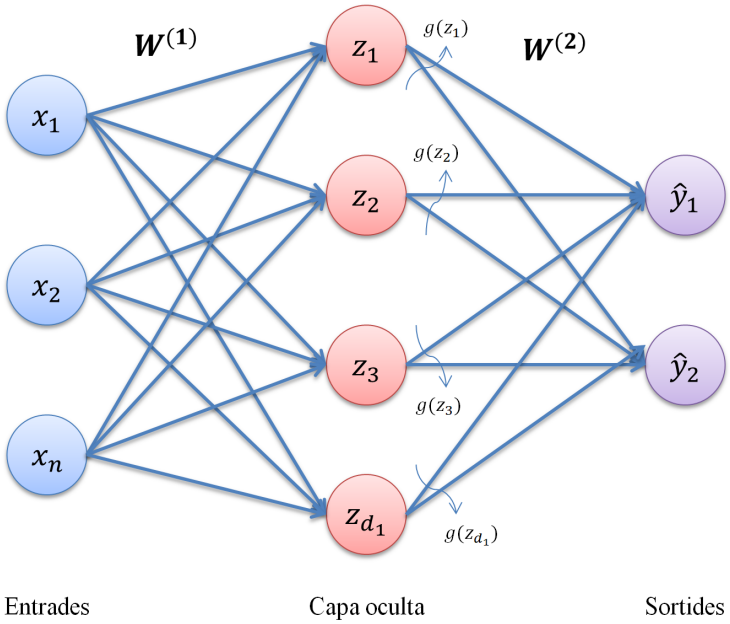


Figura 2.6: Representació gràfica d'un perceptró multicapa.

2.3.3 Tipus d'errors

L'error d'una xarxa neuronal és la diferència entre la dada de sortida de la predicció realitzada i la real; és a dir, defineix quant d'incorrecte és la predicció. Quanta més diferència hi hagi entre ambdós valors, més gran serà l'error.

Ara bé, si en comptes de tenir una dada de sortida es té una base de dades amb múltiples resultats, per saber si un model és correcte o incorrecte s'haurà de tenir en compte la mitjana aritmètica dels errors de totes les prediccions. Aquest error s'anomena error empíric.

D'altra banda, en el cas d'un model de classificació binaria on hi ha dues sortides diferents possibles, com per exemple veritat o fals, pot utilitzar-se l'error d'entropia creuada. L'entropia creuada entre dues distribucions de probabilitat mesura la diferència entre la distribució de probabilitat real i la de la predicció.

Finalment, l'error quadràtic mig s'utilitza en aquells models on les dades de sortida són números reals (models de regressió). Aquest error s'obté a partir de la mitjana dels errors al quadrat [1].

2.3.4 Mètodes d'entrenament: Retropropagació

La retropropagació (en anglés, *backpropagation*) és un dels algoritmes que s'utilitzen per entrenar eficaçment una xarxa neuronal. Aquest algoritme calcula el gradient descendent de la funció neuronal respecte als pesos determinats, sent així una aplicació pràctica de la regla de la cadena per derivades.

És important tenir en compte que en crear una xarxa neuronal s'inicialitzen els valors dels pesos aleatòriament. Per tant, el més probable és que els resultats obtinguts mitjançant aquesta xarxa no siguin

correctes i l'error de la funció sigui elevat. L'objectiu principal de l'entrenament és minimitzar l'error de la funció, ajustant els pesos i les constants a mesura que s'executa cada iteració. És precisament el gradient estocàstic descendent el que ens indica en quina direcció ha de variar (positiva o negativa) per aconseguir minimitzar aquest error [19].

2.3.4.1 Taxa d'aprenentatge

El gradient descendent indica la direcció en la que han de variar els pesos de la funció, però no sempre indica el valor o la magnitud d'aquesta variació. La taxa d'aprenentatge és el paràmetre que s'utilitza per regular la magnitud del gradient i, per tant, la variació dels pesos.

A l'hora de minimitzar una funció pot ser el cas que hi hagi mínims locals a més a més del mínim absolut, que és el que es desitja obtenir. Si la taxa d'aprenentatge es fixa a un valor massa petit, cap la possibilitat que el model convergeixi en un dels mínims locals i no aconsegueixi arribar al mínim absolut.

D'altra banda, si es selecciona una taxa massa elevada el model es saltarà el mínim absolut i no convergirà; com a conseqüència d'això, el model fallarà sense haver minimitzat l'error.

Per tant, per optimitzar l'error d'una funció s'ha de determinar una taxa d'aprenentatge suficientment gran com per evitar convergir en mínims locals, però suficientment petita com per arribar no divergir i arribar a convergir en el mínim absolut [1].

2.3.4.2 Sobreajustament

En entrenar un model, les bases de dades es divideixen entre dades d'entrenament i dades de prova o d'avaluació. Normalment, d'aquestes bases de dades s'assignen entre un 70% i un 80% de les dades per crear el model i la resta per avaluar el seu comportament.

L'objectiu principal en el Deep Learning és aprendre i generalitzar els patrons de les dades d'entrenament per obtenir un model capaç d'actuar correctament davant les dades d'avaluació, ja que són equivalents a les dades desconegudes pel model.

El subajustament succeeix quan un model és incapaç de representar tant les dades d'entrenament com les d'avaluació, resultant així una mala actuació davant les dades.

Inversament, el sobreajustament comporta un model massa complex, amb massa paràmetres, que no es capaç de generalitzar correctament davant de noves dades (dades d'avaluació). L'error en l'etapa d'avaluació, doncs, és molt elevat [20].

Regularització

Hi ha diverses tècniques per evitar el sobreajustament en un model [1, 20]:

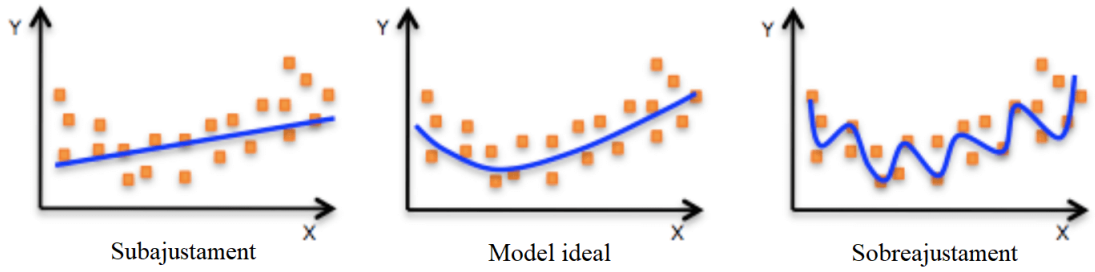


Figura 2.7: Exemple de model subajustat, ideal i sobreajustat. Font (modificada): [21]

1. *Dropout*:

   Un dels mètodes de regularització més comuns i utilitzats és el *dropout*. Aquesta tècnica consisteix en assignar aleatòriament el valor nul a alguns dels pesos de la xarxa neuronal (veure Figura 2.6), de manera que en cada iteració algunes de les neurones de la xarxa queden desactivades, obtenint així diferents resultats. Amb aquest mètode es «força» al model a buscar diferents camins de la xarxa per obtenir els resultats adequats, evitant així dependre d'un únic camí.

2. Parada a temps:

   Quan un model està entrenant-se arriba un punt on l'error d'avaluació comença a incrementar, senyal de que el model pot sobreajustar-se. Aquesta tècnica consisteix en deixar d'entrenar el model just abans de que arribi aquest punt, evitant així el problema (veure Figura 2.8).

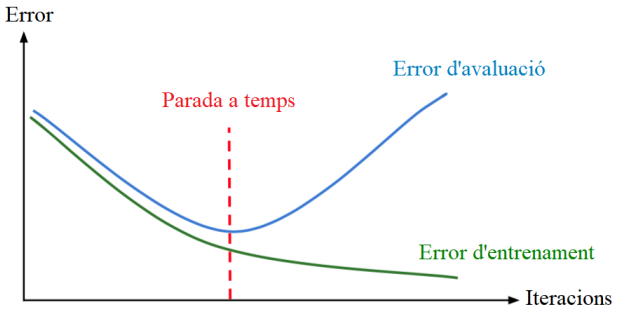


Figura 2.8: Mètodes de regularització: Parada a temps. Font (modificada): [20]

2.3.5 *Convolutional Neural Networks* (CNN)

Les CNN o ConvNets són un tipus de xarxa neuronal capaces de processar dades formades per múltiples matrius, com per exemple una imatge en color composta per tres matrius de dues dimensions (una per cada color RGB). Aquestes xarxes s'encarreguen de reduïr les images perquè siguin més fàcils de processar, però sense perdre aquelles característiques essencials per obtenir una bona predicció [22, 23].

L'estructura d'una CNN està formada per una sèrie d'etapes. Les primeres etapes estan compostes per dos tipus de capes: convolucionals i de reducció de mostreig; a l'última etapa, en canvi, acostuma a haver una capa de classificació.

2.3.5.1 Filtres

Els primers processaments d'imatges es basaven en filtres que permitien operacions com per exemple la detecció i obtenció de les vores d'un objecte situat en una imatge, a partir de combinacions de filtres de vores verticals (FVV) i horitzontals (FVH). Un filtre és una matriu la qual segons els valors assignats realitza una operació determinada. El filtre de vores verticals es defineix matemàticament amb la següent matriu:

$$FVV = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = {}^{T}FVH \tag{2.4}$$

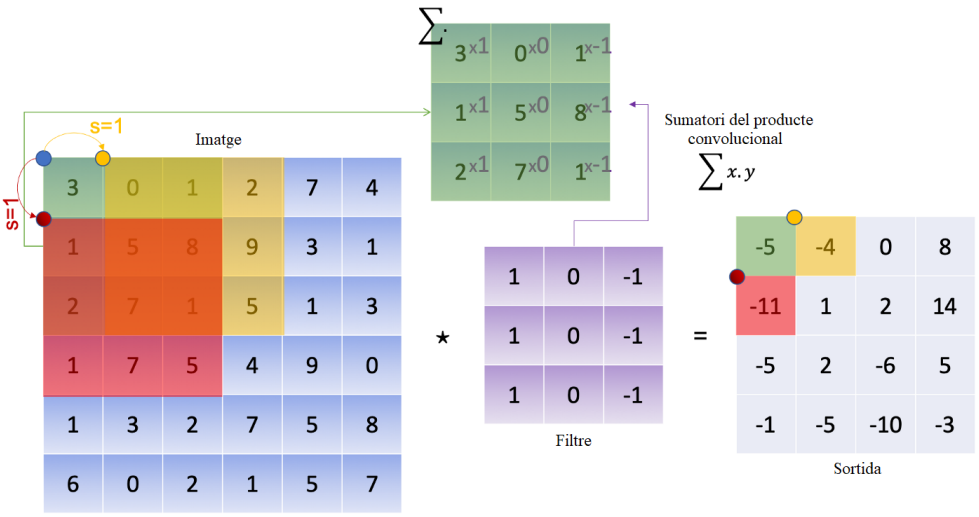


Figura 2.9: Representació gràfica del producte convolucional. Font (modificada): [24]

Aquest filtres s'apliquen a través del producte convolucional (veure Figura 2.9). Situant el filtre sobre la imatge, es realitza un producte entre les cel·les coincidents i un sumatori dels resultats obtinguts, sent aquest valor la sortida de la nova imatge en la posició del filtre (es redueix la matriu de la imatge original) [24].

2.3.5.2 Operacions bàsiques

Abans d'endinsar-se en els diferents tipus de capes, es definiran dos operacions bàsiques en aquestes xarxes neuronals: l'encoixinat (en anglés, *padding*) i el pas (en anglés, *stride*) [24].

Encoixinat

Anteriorment s'ha vist com els filtres redueixen el tamany de la imatge inicial. En utilitzar menys els pixels de les vores de la imatge en comparació amb els de l'interior, la informació d'aquestes vores es perd. Per solucionar aquest problema s'afegeix un encoixinat al voltat de la imatge, creant una nova vora de tal manera que la informació de les antigues vores és tinguda en compte. Aquesta nova vora està formada per valors nuls i pot tenir més o menys grossor depenent del valor assignat al paràmetre d'encoixinat p.

Pas

El pas és la distància que recorre el filtre (vertical i horitzontalment) després de cada producte convolucional. Quant major sigui el pas, major serà la reducció de la imatge de sortida, i a l'inversa. El paràmetre del pas s'assigna amb la lletra s (en la Figura 2.9 es pot observar un pas de $s=1$).

2.3.5.3 Capes convolucionals

Les capes convolucionals tenen la funció principal de detectar combinacions de característiques de la capa anterior. La primera capa normalment detecta característiques bàsiques; és a dir, línies verticals, horitzontals i diagonals. Aquestes característiques bàsiques es converteixen en les dades d'entrada de la següent capa, la qual obté característiques més complexes (combinacions de vores, cantonades...).

A mesura que augmentem el número de capes, aquestes comencen a detectar característiques com objectes i cares, entre d'altres [25].

Després de cada capa convolucional, generalment s'afegeix una funció d'activació per realitzar un mapeig casual no-lineal. La funció més utilitzada per aquest tipus de xarxes és l'anomenada ReLu (de l'anglés, *Rectifier Linear Unit*) i consisteix en una funció $f(x) = max(0,x)$ [26].

2.3.5.4 Capes de reducció de mostreig

Les capes de reducció de mostreig són les responsables de reduïr el tamany espacial de les característiques de les capes convolucionals i, com a conseqüència, disminuir les potència necessària per processar aquestes dades (també anomenat reducció dimensional). A més a més, aquestes capes són útils per extreure característiques dominants que no varien ni posicionalment ni rotacionalment, agilitzant d'aquesta manera el procés d'entrenament del model.

Hi ha dos mètodes de reducció de mostreig: per valor màxim o per valor promig. La reducció per valor màxim obté com a resultat el valor màxim de la porció del filtre. D'altra banda, la reducció per valor promig obté com a resultat la mitjana aritmètica de tots els valors dins la porció d'àrea corresponent. Generalment, el mètode per valor màxim obté millors resultats que per valor promig [23].

2.3.5.5 Capa de classificació

La capa de classificació rep les dades de l'última capa convolucional amb les característiques d'alta complexitat i assigna un valor de sortida comprès entre 0 i 1 que equival a la probabilitat que aquestes dades pertanyin a una classe o grup. En el cas d'una imatge, aquesta sortida ens indicaria la probabilitat que un objecte, animal, persona... es trobés a la imatge [25].

2.3.6 *Recurrent Neural Networks* (RNN)

Una RNN és una xarxa neuronal que utilitza dades de manera seqüencial; és a dir, processa les dades d'entrada d'una en una. Aquest tipus de xarxa manté un estat intern dins les capes ocultes (l'estat h_t, veure Figura 2.10) sobre els elements anteriors de la seqüència i s'actualitza a mesura que aquesta avança. Majoritàriament, aquests algoritmes s'utilitzen en problemes temporals, com per exemple traduir d'un llenguatge a un altre o reconeixement de discursos (transcriure un diàleg a text) [22].

En la Figura 2.10 es pot observar la representació gràfica desenvolupada d'una RNN. En aquesta xarxa hi trobem tres matrius de pesos:

- W_{xh}: és la matriu de pesos que transforma les dades d'entrada en cada estat de la xarxa neuronal.
- W_{hh}: és la matriu de pesos que defineix la relació entre l'estat ocult anterior i l'actual.
- W_{yh}: és la matriu de pesos que transforma l'estat ocult i dóna com a resultat les dades de sortida.

És important tenir en compte que en cadascun dels estats de la xarxa s'utilitzen les mateixes matrius de pesos.

Prèviament al subapartat 2.3.3 s'ha definit l'error com la diferència entre les dades de sortida i les d'entrada. En una RNN, per cada estat s'obté una sortida $\hat{y}_t$ que comporta un error respecte a les dades d'entrada. Per determinar l'error del model o la xarxa, doncs, s'ha de realitzar un sumatori de l'error de cada estat. Aquest error total serà l'utilitzat en l'entrenament de la xarxa [27].

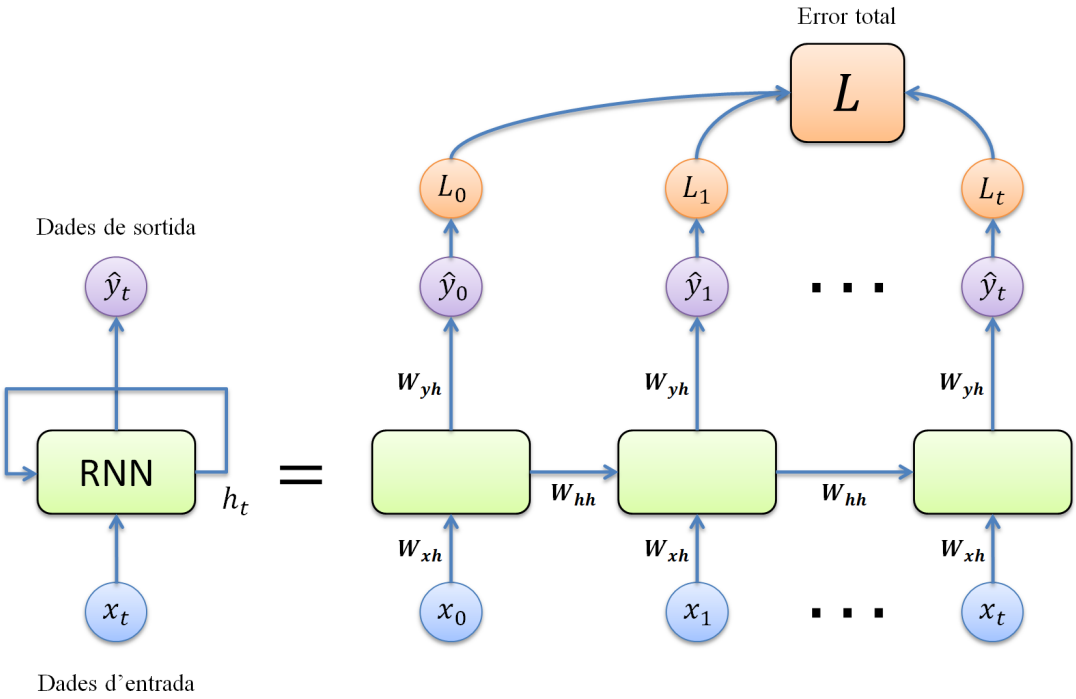


Figura 2.10: Representació gràfica de l'estructura d'una RNN tradicional.

2.3.6.1 Tipus de RNN

Les RNN es poden classificar en quatre tipus segons el mètode d'operació (veure Taula 2.1):

Taula 2.1: Classificació de les RNN. Font: [27]

Tipus de RNN	Mètode d'operació	Aplicacions
One to one	Una única entrada estàtica genera una sortida estàtica	Classificació binària
One to many	A partir d'una entrada estàtica s'obté una sortida seqüencial	Generació de text i de peus d'imatge
Many to one	A partir d'una entrada seqüencial es genera una sortida estàtica	Classificació qualitativa
Many to many	Des d'una entrada seqüencial s'obté una sortida seqüencial	Generació de música

A la Figura 2.11 es poden observar les estructures dels diferents tipus de RNN. Els rectangles vermells representen els vectors o les dades d'entrada; els de color verd, els estats interns de la xarxa neuronal; i els rectangles de color blau, els resultats o les dades de sortida.

2.3.6.2 Retropropagació a través del temps (RPTT)

La retropropagació a través del temps és l'aplicació de la retropropagació (definida a la secció 2.3.4) en les xarxes de tipus RNN. El funcionament d'aquest mètode és el mateix que en la retropropagació; el model s'entrena a si mateix calculant els errors de cada resultat o sortida i ajustant els paràmetres del model per així minimitzar l'error total [29].

En aquest cas, però, l'error de la xarxa és l'acumulat de cada estat individual (veure Figura 2.10). Per aquest motiu, en una RNN cada error individual es retropropaga des de l'error total fins al seu estat

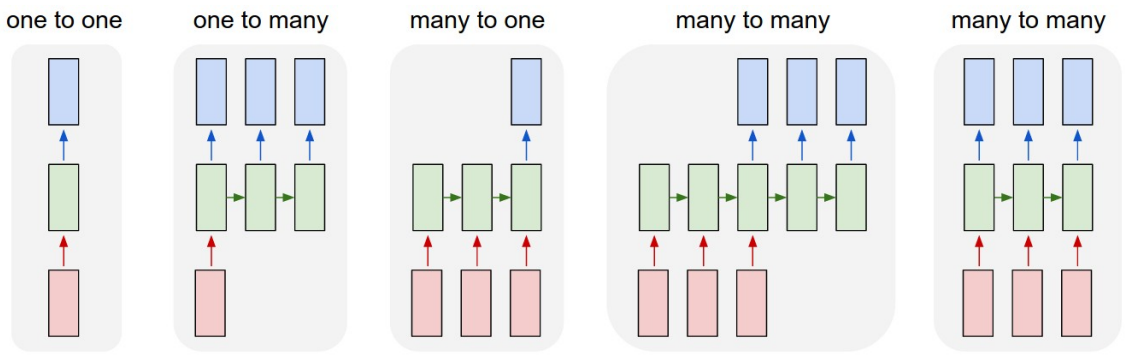


Figura 2.11: Representació gràfica dels tipus de RNN. Font: [28]

corresponent, i cada estat individual es retropropaga cap a estats anteriors en el temps (d'aquí el nom del mètode) [27].

Durant el procés de la RPTT, els models acostumen a trobar-se dos problemes: l'explosió o la desaparició dels gradients. Aquests problemes estan relacionats amb el tamany del gradient (taxa d'aprenentatge). En calcular el gradient respecte l'estat inicial h_0, aquest depèn de totes les matrius de pesos i els gradients dels estats posteriors. Quan ells gradients són massa petits, en multiplicar-se els uns amb els altres el gradient cada cop es fa més petit fins que els paràmetres dels pesos són insignificants i, per tant, l'algoritme deixa d'aprendre. Aquest esdeveniment s'anomena desaparició dels gradients.

D'altra banda, quan els gradients són massa grans succeeix just el contrari. Els paràmetres dels pesos augmenten tant que el model es torna inestable i no és capaç d'aprendre. És el que es coneix com a explosió dels gradients [29].

Existeixen diverses tècniques per evitar l'explosió o la desaparició dels gradients, com per exemple seleccionant adequadament la funció d'activació de la xarxa. Normalment s'utilitza una ReLU com a funció d'activació, ja que la seva derivada és 1 per valors positius de x. D'aquesta manera s'evita que el gradient cada cop sigui més petit i, per tant, no apareix el fenomen de la desaparició del gradient [30].

Una altra tècnica, en aquest cas per evitar l'explosió dels gradients, és inicialitzar els paràmetres dels pesos de la xarxa neuronal. Inicialitzant els pesos a la matriu d'identitat s'aconsegueix que no es redueixin a zero i siguin insignificants davant el gradient [27].

2.4 Aplicacions: conducció autònoma de vehicles

De les diverses aplicacions del Deep Learning que existeixen actualment, una de les més interessants, i en la que es centra aquest treball, és la conducció autònoma de vehicles. Dins la conducció autònoma, el Deep Learning s'utilitza en diverses funcions del vehicle, com per exemple les que es tractaran a continuació: detecció d'objectes i predicció de trajectories.

2.4.1 Segmentació semàntica

La conducció autònoma és una tasca molt complexa que depén de la percepció, el plantejament i les execucions en un ambient en constant evolució, en constant moviment. La segmentació semàntica proporciona la informació necessària per al plantejament de decisions i les execucions del vehicle, com per exemple l'espai disponible en la carretera i la detecció de carrils o senyals de trànsit.

La segmentació semàntica és una aplicació de les CNN (secció 2.3.5) que es classifica en diferents grups segons els nivells de definició de les imatges analitzades [31]:

1. Classificació d'imatge: és el bloc fonamental dins la segmentació semàntica. A partir d'una imatge en la qual només hi ha un objecte, l'ordinador és capaç de proporcionar una sortida amb l'etiqueta o paraula que representa.
2. Classificació i localització: a més a més d'etiquetar l'objecte com en el cas anterior, en aquest nivell de definició es localitza exactament la posició de l'objecte en la imatge (assumint un únic objecte per imatge). Aquesta posició és normalment indicada amb un requadre delimitador.
3. Detecció d'objectes: aquest nivell és l'ampliació de la classificació i localització per múltiples objectes dins d'una única imatge. La posició també és indicada amb un requadre delimitador.
4. Segmentació semàntica: es tracta de classificar cada píxel d'una imatge amb el grup corresponent de l'objecte que representa. A diferència dels nivells anteriors, la posició de l'objecte ja no és indicada amb un requadre delimitador, sinó pel conjunt de píxels que formen els objectes d'un mateix grup.
5. Segmentació d'instàncies: és el nivell més elevat i complex de la classificació d'objectes. Es tracta de classificar els objectes a partir dels seus píxels, amb la diferència que en aquest nivell sí es classifiquen els objectes d'un mateix grup per separat.

Figura 2.12: Classificació d'objectes segons el nivell de definició de la imatge. Font (modificada): [31]

2.4.2 Predicció de trajectòries

En la conducció autònoma, el vehicle no només ha de ser conscient de la posició o localització dels objectes que l'envolten (segmentació semàntica), sinó que també dels moviments de cadascun dels objectes. Mitjançant xarxes neuronals de tipus RNN (secció 2.3.6) es poden obtenir prediccions de les trajectòries que seguiran els objectes en moviment per així poder evitar un possible impacte [27].

3 Metodologia

Per tal de desenvolupar la tesi present, s'ha seguit la següent metodologia (resumida a la Figura 3.1 com a diagrama de blocs). Primer de tot, s'ha dut a terme una revisió de l'estat de l'art. Aquest pas consisteix en una introducció al llenguatge de programació Python, així com al Deep Learning i les diferents tècniques d'aprenentatge supervisat.

Amb els fonaments bàsics de Deep Learning i Python adquirits, el següent pas és l'adquisició i muntatge del vehicle RC que es desitja preparar perquè aprengui a moure's en un circuit tancat, en aquest cas un DonkeyCar.

Un cop muntat es procedeix a la instal·lació del *software* necessari per executar l'entrenament dels models de conducció. Aquest simplement és un ordinador virtual que utilitza un sistema operatiu Ubuntu en el qual s'instal·len els codis Python i les llibreries necessàries en funció del vehicle RC escollit. Juntament amb el software de l'ordinador, s'ha d'instal·lar el software de la unitat computacional enquestada en el vehicle, la Raspberry Pi. La instal·lació consta de dues parts: sistema operatiu (Raspbian) i codis i llibreries Python.

A continuació, el següent pas és la configuració del vehicle. Els paràmetres equivalents als valors PWM que rep el servomotor han de ser cal·librats en funció a l'angle de gir desitjat. Similar succeeix amb la cal·libració dels valors PWM que reben els motors DC, equivalents a l'empenta proporcionada. En aquest apartat també es configuren els paràmetres relacionats amb l'entrenament: taxa d'aprenentatge, percentatge de dades utilitzades en l'entrenament, etc.

En la següent fase es posa en marxa el vehicle, operant-lo manualment per tal d'obtenir les dades necessàries per l'entrenament dels algoritmes. A més a més de l'obtenció de les dades, s'ha de realitzar una preparació d'aquestes, principalment un etiquetatge de les sortides.

Tot seguit, amb un percentatge de les dades obtingudes en la conducció s'entrenen diferents models, cadascun referent a un experiment (sentit de la circulació, entrades de fora de pista, velocitat del vehicle, etc.). Després, amb el nombre restant de dades s'avalua la validesa dels models. En cas de no ser un model vàlid, es torna a la configuració del vehicle per modificar els paràmetres d'entrenament o bé es recol·lecten noves dades per realitzar un filtratge i descartar les que tinguin valors atípics o siguin errònies.

Finalment, una vegada obtinguts un o més models vàlids, es realitza una representació dels resultats, com poden ser gràfiques o valors resultants de paràmetres.

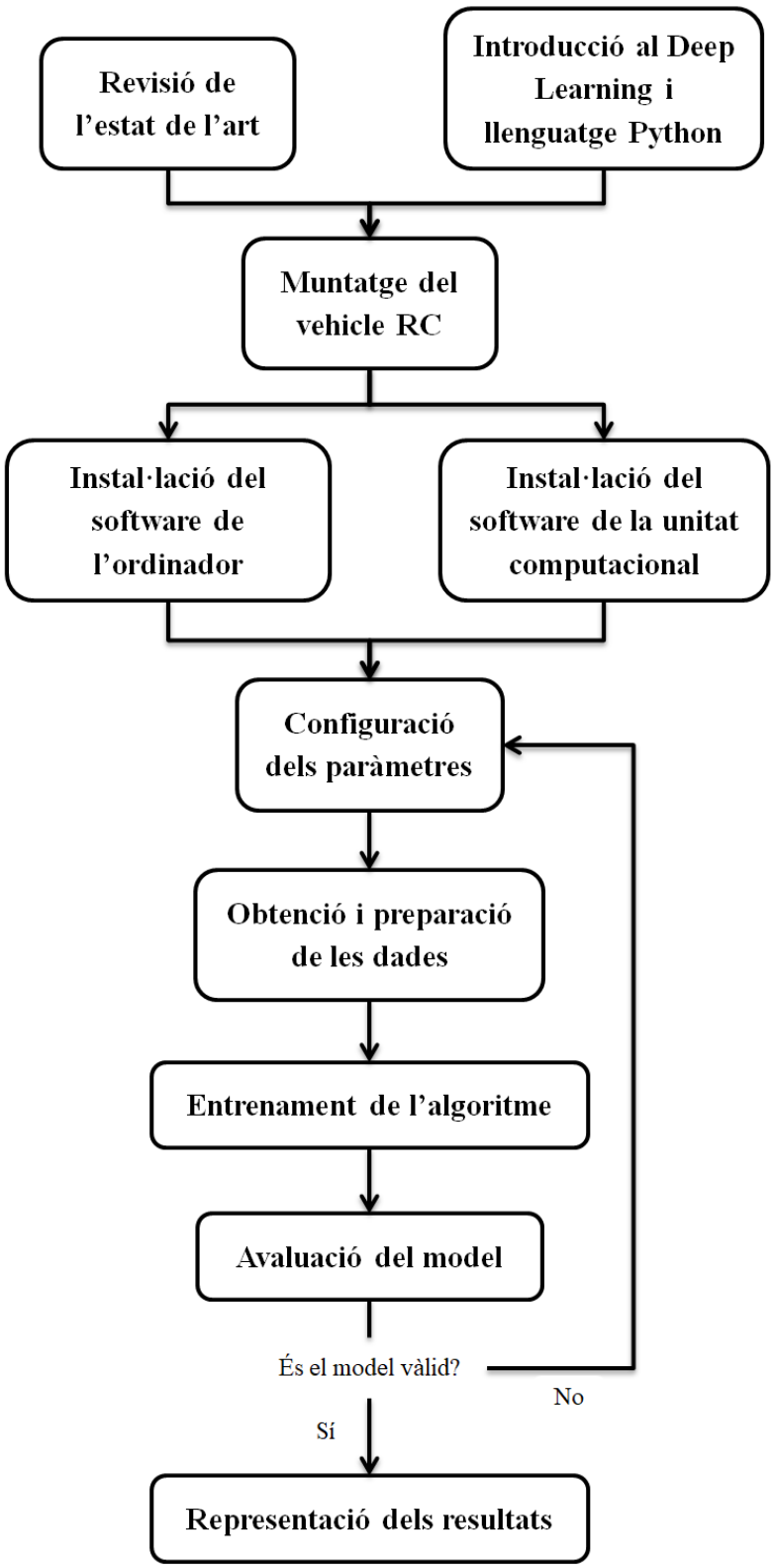


Figure 3.1: Diagrama de blocs de la metodologia del projecte.

4 Desenvolupament del DonkeyCar

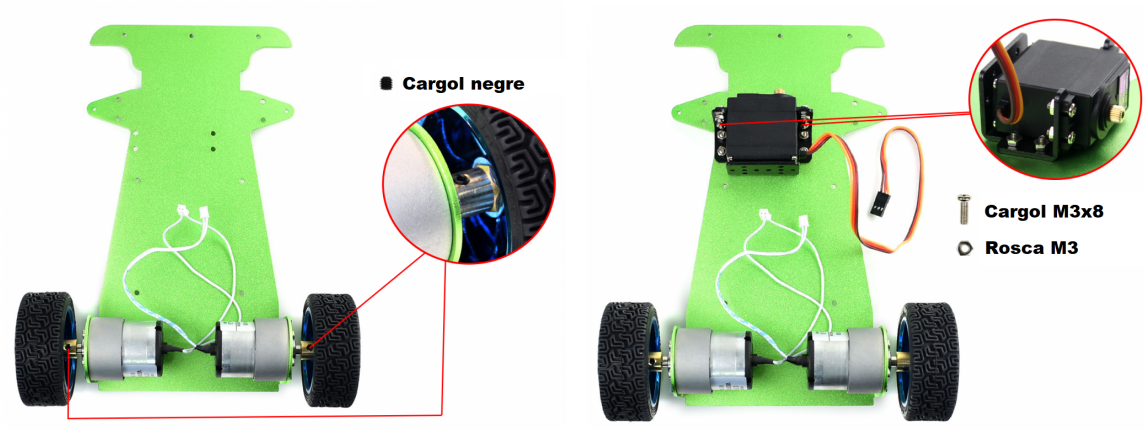
4.1 Muntatge de l'estructura mòbil

En adquirir el DonkeyCar de *Waveshare*, el seu fabricant, l'estructura mòbil d'aquest ha de ser muntada a partir de les diferents peces. Per tal de muntar correctament el vehicle, es seguirà la guia de muntatge proporcionada pel fabricant [32].

El primer pas és fixar els dos motors DC al xassís metàl·lic de color verd amb els cargols de tamany M3x6 (on M3 equival al diàmetre del cargol i 6 la longitud, ambdues mesures en mil·límetres). És important no utilitzar els cargols de tamany M3x8, ja que tot i ser només uns mil·límetres més llargs, aquests no permetrien al motor DC funcionar correctament.

A continuació, un cop muntats els acopladors en les rodes traseres i fixats amb els cargols de tamany M4x8, s'introdueixen en els eixos dels motors DC i es fixen amb els cargols negres (veure Figura 4.1a).

Després, amb els cargols de tamany M3x8 i les rosques M3, es munta el suport del servomotor juntament amb el propi servomotor, situant l'eix del servomotor cap a la part davantera del xassís (veure Figura 4.1b).



(a) Muntatge de les rodes traseres.

(b) Muntatge del servomotor.

Figura 4.1: Muntatge de les rodes traseres i el servomotor. Font (modificada): [32]

Amb l'eix traser i els servomotors fixats, és moment de muntar les barres que exerceixen la força sobre el servomotor i les rodes direccionals.

La barra del servomotor està formada per una barra metàl·lica curta i dues juntes de bola, una plana i una esfèrica (les juntes de bola es troben situades perpendicularment entre si). A més a més de la barra del servomotor, s'ha de fixar la roda del servomotor amb el suport de la mateixa, deixant el relleu d'aquest cap a fora. Finalment, s'uneix la barra amb la roda del servomotor introduint un cargol M2,5x12 juntament amb la seva rosca per la junta de bola plana.

En el muntatge de la barra direccional es necessiten dues juntes de bola, la barra metàl·lica llarga i dos artells amb els seus coixinets de rodolament corresponents.

Finalment, el conjunt de peces consta d'un artell sota la junta dreta de la barra direccional (units per un cargol M2,5x16 i la rosca corresponent) i un artell sota la junta esquerra de la barra direccional i la barra del servomotor (units per un cargol M2,5x20 i la rosca corresponent). Aquest conjunt s'uneix a les rodes davanteres introduint un cargol M4x20 i una rosca entre l'artell i la roda de cada costat (veure Figura 4.2).

Figura 4.2: Muntatge del conjunt de les barres direccionals. Font (modificada): [32]

Per unir les rodes direccionals amb el xassís del vehicle s'utilitzen separadors que es cargolen en ambdues peces amb els cargols adequats. També s'ha d'unir la roda del servomotor amb el propi servomotor. D'altra banda, els separadors s'encarreguen de subjectar tant el paraxocs com la placa d'expansió, aquesta última amb el porta bateries inclòs a la part inferior.

El suport de la càmera, la antena i la Raspberry Pi es fixen en la placa d'expansió. A més a més, en ella es connecten els cables dels motors, el cable d'alimentació i senyal del servomotor i les bateries (veure Figura 4.3).

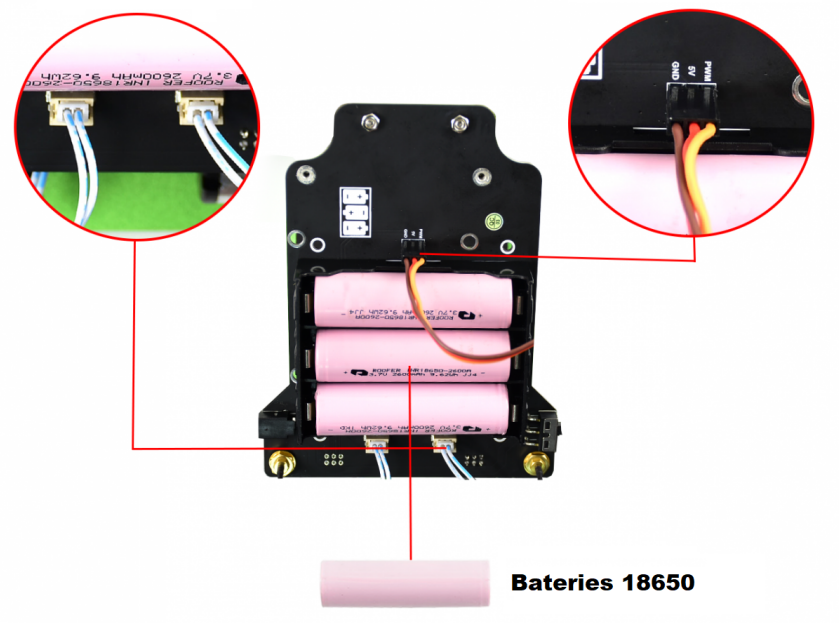


Figura 4.3: Connexió dels cables dels motors i les bateries. Font (modificada): [32]

En acabar, el xassís superior és muntat, així com la càmera en el seu suport. Una vegada la Raspberry Pi es connecta a la placa d'expansió a través del cable corresponent, el muntatge de l'estructura mòbil ha finalitzat (veure Figura 1.1).

4.2 Instal·lació del software

4.2.1 Ordinador anfitrió

L'ordinador anfitrió serà l'encarregat d'executar tot l'entrenament supervisat i crear els models de conducció autònoma. Per tal de realitzar l'entrenament cal instal·lar un ordinador virtual i, dins d'aquest, crear i instal·lar els directoris i els arxius necessaris.

En aquest projecte s'utilitzarà *Ubuntu 18.04 LTS*, un ordinador virtual amb sistema operatiu Linux. Un cop instal·lat, amb la guia de *Waveshare* com a referència es seguiran els següents passos [33]:

1. Instal·lació de miniconda Python, una eina que permet treballar amb versions diferents de cada llibreria.
2. Creació del directori pel projecte.
3. Instal·lació dels codis Python del DonkeyCar (obtinguts a *Github* [7]).
4. Creació de l'entorn anaconda.
5. Instal·lació del TensorFlow, una plataforma destinada a l'aprenentatge automàtic.
6. Creació del directori DonkeyCar.

Cada vegada que es desitgi entrar a l'entorn del DonkeyCar s'haurà d'obrir una terminal i executar la següent comanda:

conda activate donkey

4.2.2 DonkeyCar PiRacer

4.2.2.1 Sistema Operatiu

El primer pas per instal·lar el software en el DonkeyCar és la instal·lació del sistema operatiu de la Raspberry Pi. La manera més senzilla és descarregant l'aplicació *Raspberry Pi Imager* [34], una eina gràfica que permet descarregar la imatge del sistema operatiu i instal·lar-la directament a la targeta SD (en anglés, *Secure Digital*) de la Raspberry Pi [35].

Un cop connectada la targeta SD a l'ordinador mitjançant un lector de targetes, es seguiran els següents passos:

1. Obrir *Raspberry Pi Imager* i seleccionar el sistema operatiu desitjat; en aquest cas, Raspberry Pi OS.
2. Seleccionar la targeta SD on es desitja instal·lar la imatge.
3. Prémer el botó «WRITE» per començar la instal·lació.

Finalitzada la descàrrega, el següent pas és configurar la connexió WiFi (en anglés, *Wireless Fidelity*) i el protocol de comunicació remota [36].

4.2.2.2 Connexió WiFi

Per tal de poder descarregar arxius dins del DonkeyCar, aquest ha d'estar connectat a una xarxa WiFi. Es crearà un arxiu en la partició de la targeta SD anomenada «boot» que s'utilitzarà per iniciar sessió en la xarxa i poder accedir així a Internet.

De vegades la partició «boot» no és visible. Per solucionar aquest problema s'ha de desconnectar el lector de la targeta de memòria i tornar-lo a connectar.

En un editor de text s'escriurà el codi de la Figura 4.4 segons el tipus de xarxa WiFi [36].

```
country=ES
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1

network={
   ssid="<nom de la xarxa>"
   psk="<contrasenya>"
}
```

(a) Xarxa WiFi amb contrasenya.

```
country=ES
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1

network={
   ssid="<nom de la xarxa>"
   key_mgmt=NONE
}
```

(b) Xarxa WiFi sense contrasenya.

Figura 4.4: Configuració de les xarxes WiFi.

4.2.2.3 Protocol de comunicació SSH

El SSH (de l'anglés, *Secure Shell*) és un protocol que facilitza les comunicacions segures entre dos sistemes, amb una estructura client/servidor que permet a l'usuari connectar-se a un anfitrió remotament [37].

Per activar el protocol SSH de la Raspberry Pi simplement es situarà un arxiu buit anomenat «ssh» dins de la partició «boot» de la targeta SD.

Tant la configuració de la connexió WiFi com l'activació del protocol SSH només s'han de realitzar la primera vegada que es configura la Raspberry Pi [36].

4.2.2.4 Client SSH

Per comunicar-se amb la Raspberry Pi des de l'ordinador anfitrió és necessari descarregar un client SSH. *PuTTY* [38] és el client utilitzat per la realització del treball, ja que és una aplicació amb una interfaç senzilla en la que només cal introduir l'adreça IP per connectar-se remotament al DonkeyCar.

Un cop introduïda l'adreça IP del dispositiu, en aquest cas la Raspberry Pi, s'ha d'iniciar sessió amb el nom d'usuari i la contrasenya [36]:

- Usuari: pi
- Contrasenya: raspberry

4.2.2.5 Configuració de la Raspberry Pi

Finalment, totes les llibreries i dependències pel funcionament del vehicle han de ser instal·lades en la Raspberry Pi. Seguint la guia de *Waveshare* [33], fabricant del DonkeyCar, s'obrirà una terminal de l'ordinador i s'executaran els següents passos:

1. Instal·lació de llibreries Python.
2. Instal·lació de llibreries per l'OpenCV.
3. Configuració de l'entorn virtual.
4. Instal·lació dels codis Python del DonkeyCar (obtinguts a *Github* [7]).
5. Instal·lació de l'OpenCV.
6. Instal·lació de la pantalla OLED (utilitzada per mostrar l'adreça IP, el voltatge i la intensitat...).
7. Creació del DonkeyCar.

4.3 Configuració del DonkeyCar

Finalitzada la instal·lació del *software* tant en l'ordinador anfitrió com en la Raspberry Pi, s'han de configurar els paràmetres del DonkeyCar per tal d'obtenir un correcte funcionament [33]. Dels codis Python obtinguts a *Github* [7], s'ha de modificar l'anomenat «myconfig.py».

Aquest codi conté tots els paràmetres relacionats amb el vehicle: la càmera, la direcció de les rodes, l'accelerador, etc.; així com els paràmetres relacionats amb l'entrenament de l'algoritme: nombre de màximes èpoques, taxa d'aprenentatge, percentatge de dades utilitzades per l'entrenament, entre d'altres.

Per editar-lo simplement s'executarà la següent comanda en la terminal del client SSH (en aquest cas, *PuTTY*):

nano ~/mycar/myconfig.py

4.3.1 Calibratge de la direcció

Per tal de calibrar aquests paràmetres s'haurà de buscar el seu canal corresponent dins del codi «myconfig.py». Després, executant:

donkey calibrate --channel <canal_direcció> --bus=1

s'accedeix al calibratge de la direcció de les rodes del vehicle.

Primer de tot s'ha de trobar el valor PWM (en anglés, *Pulse Width Modulation*) que situa les rodes rectes. En aquest cas, a partir de proves de diferents valors s'obté un valor final de 550.

A continuació s'han d'obtenir els valors PWM que fan girar les rodes completament en ambdues direccions. Seguint el mateix procediment del cas anterior s'aconsegueixen uns valors de 390 en gir màxim cap a l'esquerra i de 670 en gir màxim cap a la dreta.

Com es pot observar, a causa de la distribució i el muntatge del servomotor en la secció 4.1, hi ha una direcció en la qual les rodes poden girar més que en l'altra. Per tal de compensar la direcció del vehicle s'han de seleccionar uns valor equidistants al valor de referència, aquell que situa les rodes rectes.

Finalment, els valors resultats obtinguts, juntament amb els paràmetres a modificar són els següents:

Taula 4.1: Paràmetres de la direcció del vehicle.

Direcció	Paràmetre	Valor PWM
Gir cap a l'esquerra	*STEERING_LEFT_PWM*	430
Rodes rectes	-	550
Gir cap a la dreta	*STEERING_RIGHT_PWM*	670

4.3.2 Calibratge de l'acceleració

El calibratge de l'acceleració és similar a la de la direcció de les rodes. El procediment a seguir és exactament el mateix: buscar el canal corresponent, accedir al calibratge, trobar els valors PWM corresponents per l'acceleració cap endavant i cap enrere i, finalment, modificar el codi de paràmetres amb els valors obtinguts.

Tanmateix, durant el calibratge és possible que per cap valor PWM s'observi una acceleració de les rodes. Per tal de solucionar aquest problema s'han de modificar les adreces dels controladors PCA9685.

El PCA9685 és un controlador de modulació per ample de pulsacions (PWM) que és controlat per I2C, un bus de comunicacions en sèrie [39]. En el DonkeyCar es poden observar dos controladors PCA9685: un encarregat de controlar el servomotor (direcció del vehicle) i l'altre encarregat de controlar els motors DC (acceleració del vehicle). En el cas d'haver un error en l'adreça dels controladors dins el codi dels paràmetres, el motor corresponent a l'adreça no funcionaria.

Executant el codi «myconfig.py», modificarem les adreces dels controladors:

- *PCA9685_I2C_ADDR = 0x40*: corresponent al servomotor de la direcció del vehicle.
- *PCA9685_I2C_ADDR1 = 0x60*: corresponent als motors de l'acceleració del vehicle.

Solucionat el problema de l'acceleració, es poden observar a la següent taula els valors dels paràmetres corresponents:

Taula 4.2: Paràmetres de l'acceleració del vehicle.

Acceleració	Paràmetre	Valor PWM
Cap endavant	*THROTTLE_FORWARD_PWM*	4095
Parat	*THROTTLE_STOPPED_PWM*	0
Cap enrere	*THROTTLE_REVERSE_PWM*	-4095

És important assignar un valor negatiu a l'acceleració cap enrere del vehicle ja que, de no ser així, els motors no serien capaços de girar en sentit contrari.

4.4 Posada en marxa

Per posar en marxa i controlar el DonkeyCar s'utilitzaran dos mètodes diferents, cadascun amb una finalitat. El primer mètode és a través d'una pàgina web o digital. Mitjançant aquest mètode es podrà comprovar que tant les direccions com l'acceleració del vehicle funcionen correctament. D'altra banda, el segon mètode és a través d'un comandament de consola, amb el qual es controlarà el vehicle per obtenir les dades necessàries per l'entrenament [33].

4.4.1 Pàgina web

Primerament, des de la terminal del client SSH s'han d'executar les següents comandes:

cd mycar

python manage.py drive

Aquesta comanda executa el codi Python corresponent a la conducció a partir dels paràmetres configurats prèviament.

Des de l'ordinador anfitrió, obrint la pàgina web «https://<adreça_ip_raspberrypi>:8887» s'observa una interfaç a través de la qual es pot controlar el vehicle. Per comprovar el correcte funcionament del DonkeyCar s'utilitzen els següents controls en el teclat:

- «I»: augmentar acceleració.

- «K»: disminuir acceleració.
- «J»: girar a l'esquerra.
- «L»: girar a la dreta.

Un cop comprovat el correcte funcionament es pot avançar a l'últim pas de la posada en marxa del vehicle.

4.4.2 Comandament

Juntament amb les peces necessàries pel muntatge de l'estructura del DonkeyCar s'inclou un comandament de consola amb el qual es pot controlar el vehicle.

Després de connectar l'adaptador USB (en anglés, *Universal Serial Bus*) del comandament a la Raspberry Pi, des de la terminal s'han d'executar les següents comandes:

cd mycar

python manage.py drive --js

Similar al cas anterior, aquesta comanda permet conduir el vehicle a través del comandament. Els controls utilitzats són els següents:

- Palanca esquerra: moure la direcció de les rodes.
- Palanca dreta: accelerar o desaccelerar el vehicle.
- «R2»: augmentar la màxima acceleració disponible.
- «L2»: disminuir la màxima acceleració disponible.
- «Quadrat»: esborrar les dades obtingudes en els últims tres segons.
- «Triangle»: parar la conducció i captació d'imatges.

Com es veurà en el següent capítol, amb l'ajut del comandament es podran obtenir les imatges emprades en l'entrenament dels models de conducció autònoma.

5 Entrenament supervisat

Anteriorment, a la secció 2.2 s'han definit les tres categories principals de *Machine Learning* que existeixen. L'aprenentatge supervisat és aquell que es caracteritza per treballar amb bases de dades etiquetades. Per tant, a l'hora d'obtenir les dades d'entrada emprades en l'entrenament del model hauran d'etiquetar-se les sortides de cadascuna.

5.1 Obtenció de dades

5.1.1 Codi d'execució: funció *drive*

A més a més de posar en marxa el vehicle, cal executar un codi que capti les imatges i les guardi en la memòria per tal de poder ser utilitzades posteriorment en l'entrenament. Aquesta funció es troba implementada en el codi anomenat «manage.py», obtingut al *Github* de *Waveshare* [7]. A continuació s'explicarà el funcionament del codi pas a pas.

Primerament, com en la majoria de codis, cal importar les llibreries necessàries (Tensorflow, Numpy...). Tot seguit, s'importen les *parts* (en aquest cas del DonkeyCar). Una *part* és una classe creada a Python que comprèn un component funcional d'un vehicle, ja siguin sensors, actuadors o controladors, entre d'altres. Per aquelles on es triga més en executar existeix el concepte de *threaded parts*, que són les que s'executen en paral·lel (o en un altre fil) i, per tant, no enredereixen el bucle d'iteració [40].

Al final de tot es pot observar el *main*, que es divideix en tres parts. Un argument d'entrada (funció docopt) a partir del qual es seleccionarà quina funció executar, *drive* o *train*; carregar els paràmetres

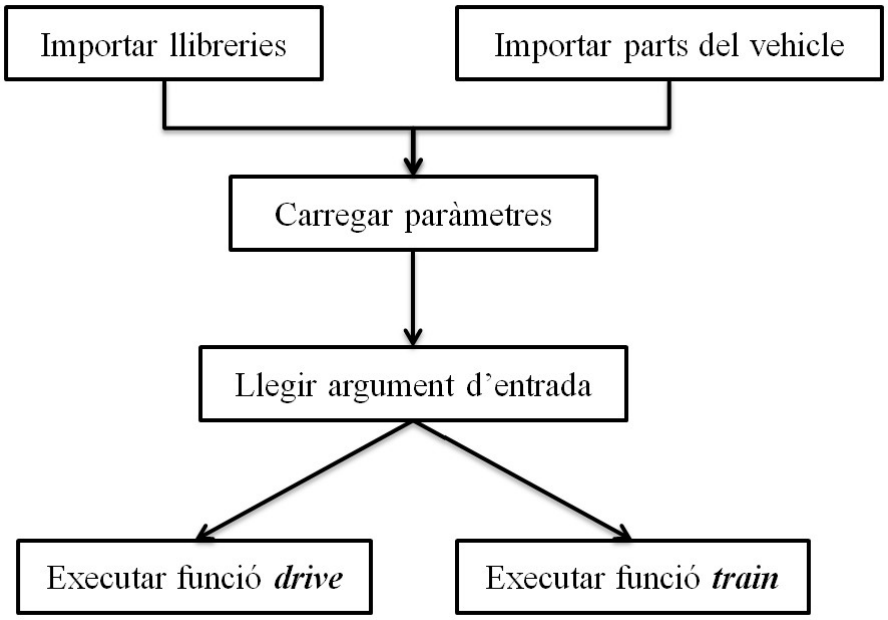


Figura 5.1: Diagrama de fluxe del codi «manage.py».

definits al codi «myconfig.py» (secció 4.3) i l'execució de la funció escollida (veure diagrama de fluxe a la Figura 5.1).

En aquest cas, en executar la funció *drive*, a través de la qual s'obtindran les dades, s'inicialitza el vehicle. Inicialitzar el vehicle equival a crear una classe que gestiona el funcionament i execució de cada *part*, així com gestionar una classe de memòria. A partir d'aquí es van creant i afegint les diferents *parts* del vehicle.

La primera *part* és la càmera. Segons el tipus de càmera definida a les configuracions, es crea una *part* on les sortides estan formades pels vectors de les imatges obtingudes, i s'afegeix a la llista de *parts* mitjançant la funció *V.add*.

A continuació, es crea i s'afegeix la *part* del comandament, ja sigui manual o per pàgina web. Aquest comandament té com a entrades els vectors de la càmera i, com a sortides, els valors de l'angle de direcció, l'acceleració, el mode de conducció i una variable que indica si s'estan grabant les imatges.

En el cas de disposar d'un model preentrenat es pot carregar mitjançant funcions de la llibreria Keras, facilitant d'aquesta manera l'entrenament i reduint el temps d'execució. A més a més de carregar el model preentrenat, la funció *kl.load* també s'utilitza per carregar el model final a l'hora d'avaluar la conducció autònoma de l'algoritme.

Com s'ha vist a la secció 4.3.2, les variables direcció i acceleració es gestionen mitjançant els controladors PCA9685. Aquestes *parts*, doncs, també han de ser afegides al vehicle per tal d'obtenir les entrades de l'angle i l'acceleració del vehicle, que seran utilitzades a continuació per etiquetar la base de dades.

Una vegada obtingudes les variables es creen les carpetes amb els arxius JSON pels valors de sortida (angle, acceleració i mode) i JPEG pels valors d'entrada (vectors de la imatge). Amb la funció *TubHandler* es podrà assignar un directori a les carpetes.

Finalment, amb la funció *V.start* es posen en marxa totes les *parts* creades del vehicle, iterant-les en l'ordre d'entrada del codi (veure diagrama de fluxe a la Figura 5.2).

5.1.2 Base de dades etiquetades

La capacitat de la bateria no permet operar el vehicle el temps suficient per adquirir les dades necessàries per l'entrenament. Per formar una base de dades etiquetades suficientment gran s'han realitzat diverses conduccions en les quals s'han captat entre 1000 i 3000 imatges de manera que, posteriorment, ajuntant-les s'obtenen un total de 9000 imatges aproximadament (juntament amb les dades de sortida pertinents). Per evitar una desconnexió del vehicle és aconsellable carregar les bateries després de cada conducció o cada 3000 imatges captades.

Al capítol 6 es desenvoluparan els diferents experiments realitzats segons el tipus de conducció. Per aquest motiu, les bases de dades estan dividides en subgrups: imatges en sentit antihorari, sentit horari, ambdos sentits i entrada a pista. En funció de l'experiment es seleccionaran les bases de dades adequades.

5.2 Entrenament dels models

5.2.1 Codi d'execució: funció *train*

Com s'ha vist prèviament, depenent l'argument d'entrada s'executa una de les dues funcions. En l'entrenament del model aquesta funció és importada d'un altre codi, «train.py». Aquest apartat explicarà el funcionament principal del codi.

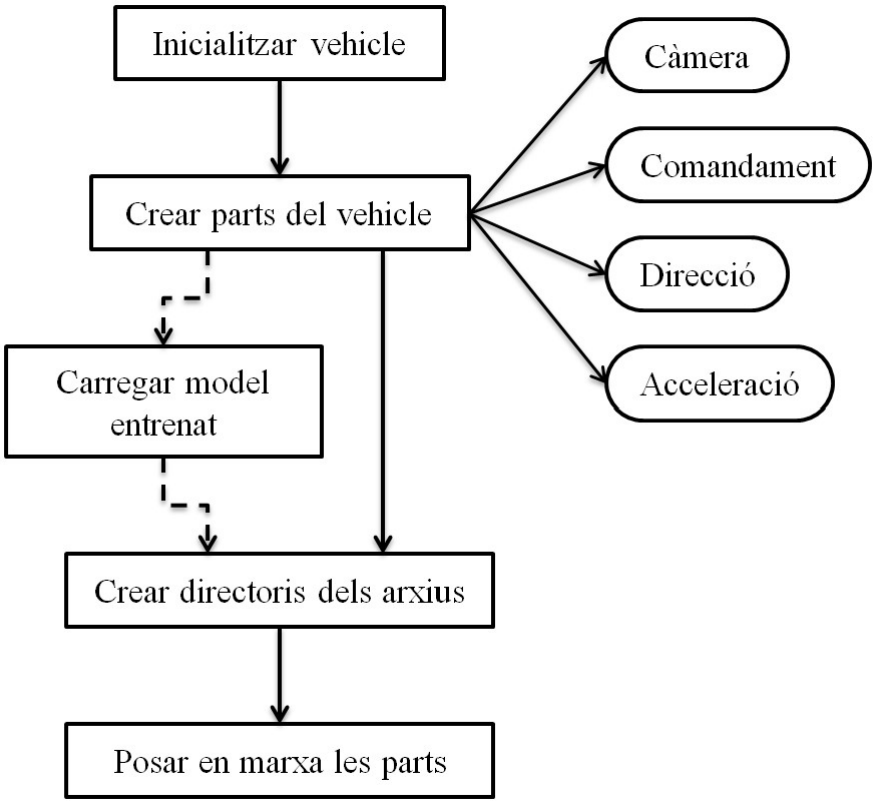


Figura 5.2: Diagrama de fluxe de la funció *drive*.

Similar al codi anterior, primer s'importen les llibreries i les *parts* del vehicle. És important destacar la llibreria Keras, ja que serà la llibreria utilitzada per l'entrenament de la xarxa neuronal.

A continuació, s'obren els directoris on es troben els arxius JSON i JPEG amb els valors de sortida i entrada, respectivament. Segons el criteri de partició escollit en la configuració del vehicle (80% entrenament, 20% avaluació) es divideixen i es barregen les dades entre si.

Amb les dades preparades s'executa les funcions del codi «keras.py», on es realitzarà l'algoritme del tipus de xarxa neuronal seleccionada.

5.2.2 Entrenament lineal

L'entrenament utilitzat en la creació de models de conducció autònoma és de tipus lineal. El *pilot* (de l'anglés, *Programmed Inquiry, Learning or Teaching*) de Keras lineal utilitza una neurona per obtenir un valor de sortida continu a través de les diferentes capes de la xarxa. En aquest cas hi ha una per la direcció i una altra per l'acceleració. A més a més, la sortida no es delimita.

Els avantatges d'aquest tipus d'entrenament són: conducció fluida (referent a la direcció del vehicle), robustesa de l'algoritme i manca de limitacions arbitràries en la direcció i acceleració. D'altra banda, l'inconvenient d'aquest mètode és que de vegades pot fallar en l'aprenentatge de l'acceleració [41].

A continuació es pot observar el codi de la xarxa neuronal a l'Algorisme 5.1, juntament amb un diagrama esquematitzat del seu funcionament (veure Figura 5.3). Es tracta d'una CNN (definida a la secció 2.3.5).

Primer s'hi troba la capa d'entrada amb les dades de les imatges (*img_in*). La CNN està formada per cinc capes convolucionals amb una funció d'activació tipus ReLU per aplicar un mapeig no-lineal.

Algorisme 5.1 Codi d'execució de l'algoritme lineal Keras.

```
def default_n_linear(num_outputs, input_shape=(120, 160, 3),
    roi_crop=(0, 0)):

    drop = 0.1

    input_shape = adjust_input_shape(input_shape, roi_crop)

    img_in = Input(shape=input_shape, name='img_in')
    x = img_in
    x = Convolution2D(24, (5,5), strides=(2,2), activation='relu',
        name="conv2d_1")(x)
    x = Dropout(drop)(x)
    x = Convolution2D(32, (5,5), strides=(2,2), activation='relu',
        name="conv2d_2")(x)
    x = Dropout(drop)(x)
    x = Convolution2D(64, (5,5), strides=(2,2), activation='relu',
        name="conv2d_3")(x)
    x = Dropout(drop)(x)
    x = Convolution2D(64, (3,3), strides=(1,1), activation='relu',
        name="conv2d_4")(x)
    x = Dropout(drop)(x)
    x = Convolution2D(64, (3,3), strides=(1,1), activation='relu',
        name="conv2d_5")(x)
    x = Dropout(drop)(x)

    x = Flatten(name='flattened')(x)
    x = Dense(100, activation='relu')(x)
    x = Dropout(drop)(x)
    x = Dense(50, activation='relu')(x)
    x = Dropout(drop)(x)

    outputs = []

    for i in range(num_outputs):
        outputs.append(Dense(1, activation='linear', name='
            n_outputs' + str(i))(x))

    model = Model(inputs=[img_in], outputs=outputs)

    return model
```

Després de cada capa convolucional es realitza un *dropout*, una tècnica de regularització que evita el sobreajustament del model assignant aleatòriament un valor nul al 10% de les neurones de la capa (veure secció 2.3.4.2).

En cada capa convolucional es poden observar tres paràmetres d'entrada. El primer valor és el nombre de neurones que té la capa oculta (veure Figura 2.6). El segon paràmetre, format per dos valors, és el tamany del filtre seleccionat. El paràmetre *stride* és el tamany del pas, definit anteriorment a la secció 2.3.5.2. Finalment, es defineix la funció d'activació.

A la sortida de la CNN es situa una altra xarxa neuronal. Primer de tot realitza la operació de *flattening*, que consisteix en transformar una matriu de dues dimensions a un vector d'una única dimensió, de tal manera que aquestes dades poden ser utilitzades com a entrades per la següent capa.

Tot seguit hi ha dues capes ocultes, amb 100 i 50 neurones respectivament, que formen diferents

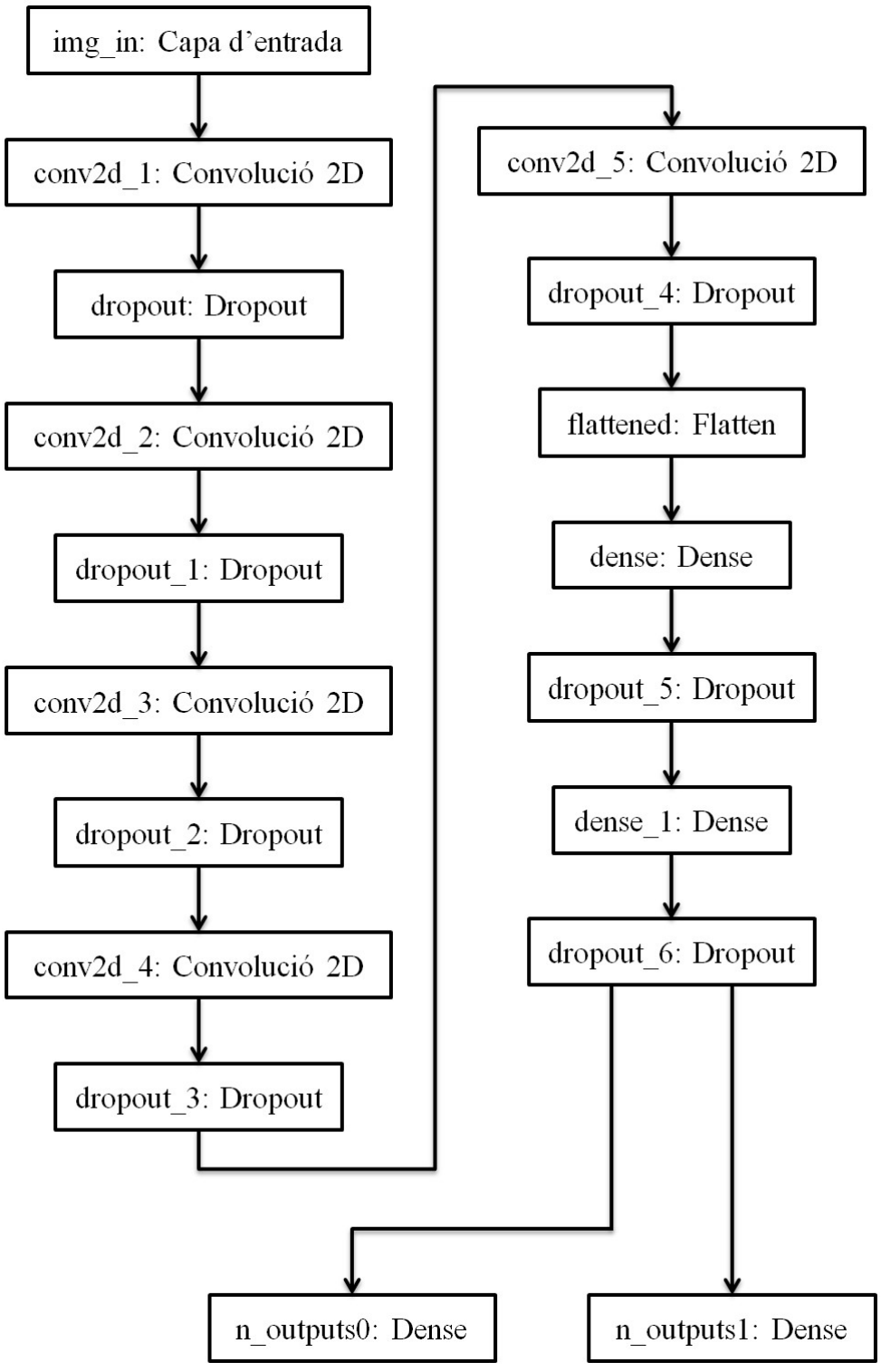


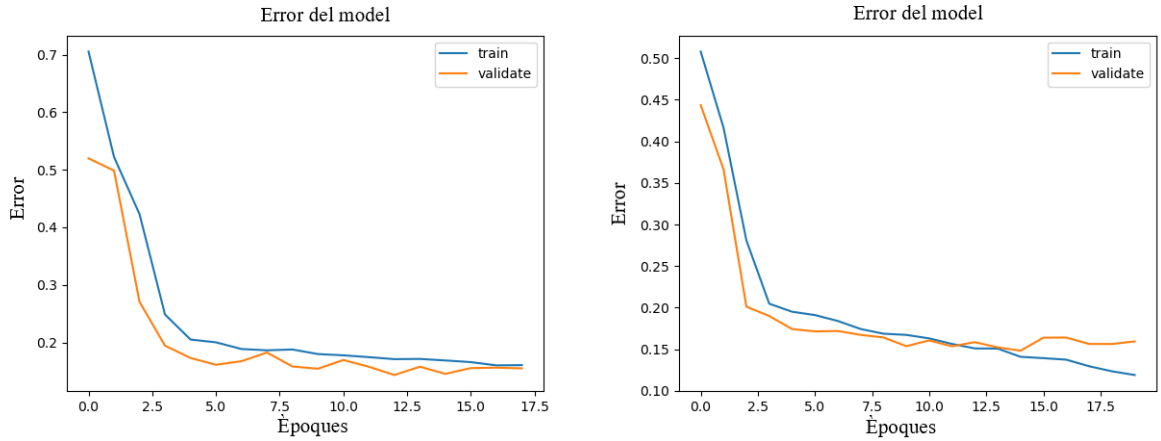
Figura 5.3: Diagrama de fluxe de l'entrenament amb Keras.

camins o possibilitats per tal de trobar finalment el camí a través del qual la sortida és més adequada. Novament, després de cadascuna de les capes ocultes es situa un *dropout* per evitar el sobreajustament del model.

Finalment, a la capa de sortida s'obté un valor final de l'angle per la direcció i un altre per l'acceleració referent a la imatge d'entrada. Amb les dades d'entrada i sortida definides, el model pot ser creat a través de la funció *Model* importada de Keras (llibreria de TensorFlow).

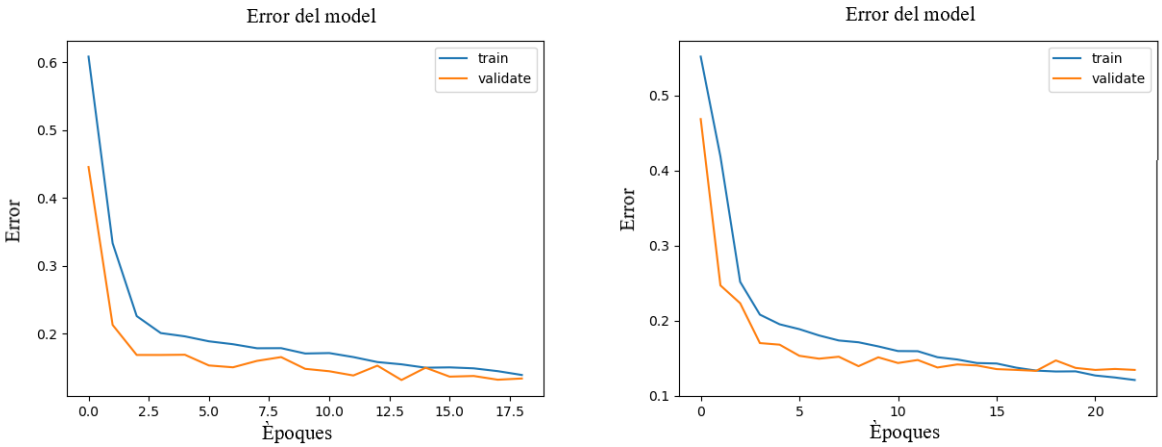
6 Anàlisi dels resultats

En aquest apartat s'analitzaran els resultats obtinguts en l'entrenament de quatre models diferents, anomenats alfabèticament. Cadascun dels experiments s'ha realitzat amb un dels següents tipus de conducció: sentit únic, ambdós sentits i ambdós sentits més entrada a pista. A continuació, a la Figura 6.1 es poden observar les gràfiques obtingudes de l'error de cada model en funció de les èpoques d'entrenament, tant per l'etapa d'entrenament (en color blau) com per la d'avaluació (en color taronja).



(a) Model A.

(b) Model B.



(c) Model C.

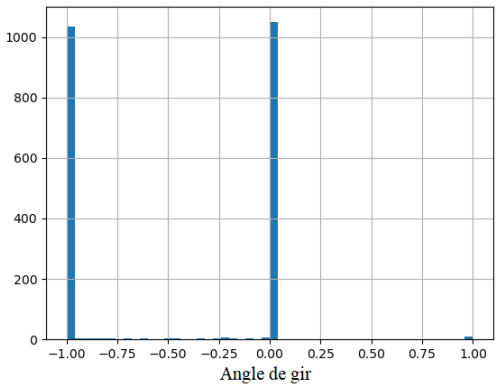
(d) Model D.

Figura 6.1: Representació gràfica de l'error del model en funció de les èpoques d'entrenament.

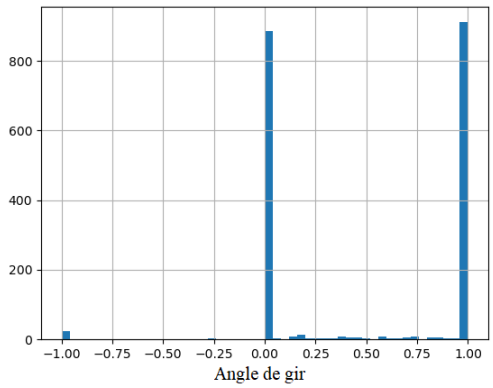
Com s'ha comentat al capítol anterior (secció 5.1.2), hi ha quatre tipus de bases de dades: sentit antihorari, sentit horari, ambdós sentits i entrada a pista. Cada experiment realitzat utilitzarà un o més tipus de bases de dades per entrenar el model.

Per saber exactament el tipus de cada base de dades es realitzarà un histograma en el qual s'indicarà el recompte de les imatges amb el mateix angle de gir. Així, observant les representacions gràfiques

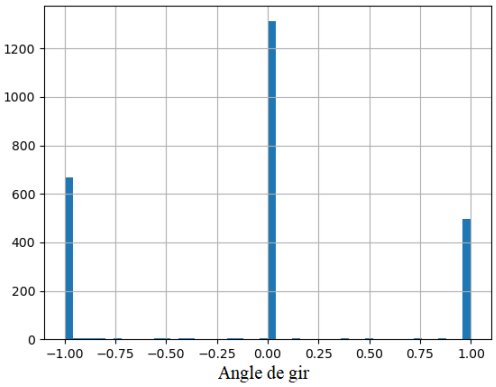
a la Figura 6.2 es pot confirmar que aquelles bases de dades on predomina l'angle de gir negatiu són de sentit antihorari; aquelles on predomina l'angle de gir positiu són de sentit horari; i finalment, aquelles on hi ha angles de gir positius i negatius, d'ambdós sentits. En el cas d'entrada a pista, en no diferenciar-se del cas d'ambdós sentits, els directoris d'aquestes bases de dades han de ser destacats d'alguna manera (per exemple, indicant-ho al nom).



(a) Base de dades en sentit antihorari.



(b) Base de dades en sentit horari.



(c) Base de dades en ambdós sentits.

Figura 6.2: Histogrames segons els tipus de bases de dades.

6.1 Experiments realitzats

6.1.1 Experiment A

El primer experiment consisteix en entrenar un model a partir de bases de dades d'un únic sentit, en aquest cas sentit horari. L'error del model resultant es representa a la Figura 6.1a. S'observa com en tot moment l'error d'avaluació (en color taronja) és inferior a l'error d'entrenament (en color blau), indici d'un bon entrenament del model. L'error final obtingut, en canvi, és alt en comparació amb els errors obtinguts en altres models. Per tant, fins avaluar el model no es pot confirmar la validesa d'aquest.

6.1.2 Experiment B

El següent experiment realitzat és un model entrenat amb bases de dades d'ambdós sentits. A la Figura 6.1b s'observa com el model respon correctament en l'etapa d'entrenament, ja que obté un

valor d'error baix. Tanmateix, a l'hora de validar les dades en l'etapa d'avaluació, l'error del model augmenta considerablement en les èpoques finals, donant-se el fenomen del sobreajustament (secció 2.3.4.2).

El resultat obtingut, doncs, és un model massa complex, amb massa paràmetres, que no és capaç d'actuar correctament davant noves dades i, per tant, un model no vàlid. Una possible solució per tal d'haver evitat aquest problema hagués estat emprar la tècnica de la parada a temps (veure Figura 2.8), deixant d'entrenar el model entre la 13a i la 14a època, on els valors d'ambdues etapes eren similars. Tot i així, l'error total en aquelles èpoques segueix sent bastant elevat com per ser considerat un model vàlid.

6.1.3 Experiment C

Com a conseqüència dels resultats obtinguts en l'experiment anterior, s'ha entrenat una altra vegada un model a partir de bases de dades de sentit horari, de sentit antihorari i d'ambdós sentits.

Sense tenir en compte les diferents conduccions en cada experiment, a primera vista els models de les Figures 6.1a i 6.1c són molt similars. Per tal de diferenciar-los s'han de definir dos conceptes nous: precisió i exactitud. La precisió es defineix com el nivell de proximitat o similitud entre els resultats de diferentes mesures entre si. D'altra banda, l'exactitud determina el nivell de proximitat d'un resultat amb el valor real [A2].

Tenint en compte aquests conceptes, ambdós models resulten ser models precisos, ja que generalitzen correctament els patrons trobats en l'etapa d'entrenament. Tanmateix, el model C resulta ser més exacte que el model A, obtenint un error menor i, per tant, sent un model més vàlid.

El motiu pel qual un model és més precís que l'altre esdevé de l'obtenció de dades (secció 5.1). La conducció realitzada durant la recaptació d'imatges determina la qualitat de les dades proporcionades al model. Si les imatges proporcionades al model no són adequades; és a dir, el model s'alimenta amb dades d'una conducció dolenta (sortint-se del carril, no girant en el moment adequat...), el model resultant serà poc precís.

A més a més de la qualitat de dades, un altre aspecte important a tenir en compte és la quantitat de dades. Si un model no té suficients dades d'entrada, és probable que l'entrenament no aconsegueixi trobar els patrons necessaris per convertir-se en un model exacte.

En aquest cas, la diferència en les dades proporcionades a ambdós models resta tant en la quantitat com en la qualitat. El model C (Figura 6.1c) ha sigut entrenat amb més dades d'entrada, amb més imatges, i durant la recaptació d'aquestes dades la conducció ha sigut òptima. A falta de l'avaluació d'aquest, doncs, es pot afirmar que és un possible model vàlid.

6.1.4 Experiment D

Finalment, l'últim experiment consisteix en entrenar un model per poder realitzar una conducció autònoma en qualsevol dels sentits del circuit i, a més a més, poder entrar-hi des de fora de pista. Els resultats obtinguts a la Figura 6.1d indiquen un bon entrenament del model, ja que els errors obtinguts no són gaire elevats i, tot i augmentar l'error d'avaluació en les últimes èpoques, l'entrenament s'ha parat a temps (a diferència de l'experiment B). Així com en el cas anterior, hi ha la possibilitat que el model obtingut sigui vàlid.

6.2 Avaluació dels models

Per comprovar la validesa dels models entrenats es sometran a dos criteris d'avaluació. Primer de tot, i el més important, es carregaran els models de conducció autònoma al vehicle i es deixarà circular

per la pista per observar el seu comportament davant noves dades. A continuació, es realitzaran prediccions del model de bases de dades d'entrada i s'analitzaran els resultats obtinguts gràficament.

6.2.1 Conducció autònoma del vehicle

Primer de tot, en carregar el model A en el DonkeyCar, el vehicle aconsegueix realitzar unes quantes voltes al circuit en sentit horari fins que perd la direcció i surt del carril. En canviar el sentit de la conducció, el vehicle ni tan sols aconsegueix completar una volta i es desvia. Repetint aquest procés dues vegades més s'obtenen resultats similars. A partir d'aquestes premisses es determina que aquest primer model no és un model vàlid per la conducció autònoma del vehicle. Aquest resultat era d'esperar, ja que durant tot l'entrenament el model A ha sigut alimentat amb una base de dades de sentit horari.

L'actuació del vehicle amb el model B carregat és molt similar a la de l'anterior, amb la diferència que en canviar de sentit de circulació sí aconsegueix completar algunes voltes, tot i acabar sortint-se del circuit. Com a conseqüència dels problemes de sobreajustament del model, aquest resulta no ser vàlid per la conducció autònoma.

Carregant el model C i posant en marxa el vehicle s'observa com aquest realitza una conducció fluïda i eficient, mantenint-se en tot moment dins del carril i girant correctament. La mateixa resposta s'obté en canviar de sentit i realitzar voltes durant uns quants minuts. El vehicle és capaç fins i tot de començar fora de pista i incorporar-se per seguir circulant per dins, tot i no haver sigut entrenat específicament per aquesta maniobra. Segons els resultats obtinguts en la conducció autònoma es pot afirmar que aquest sí és un model vàlid.

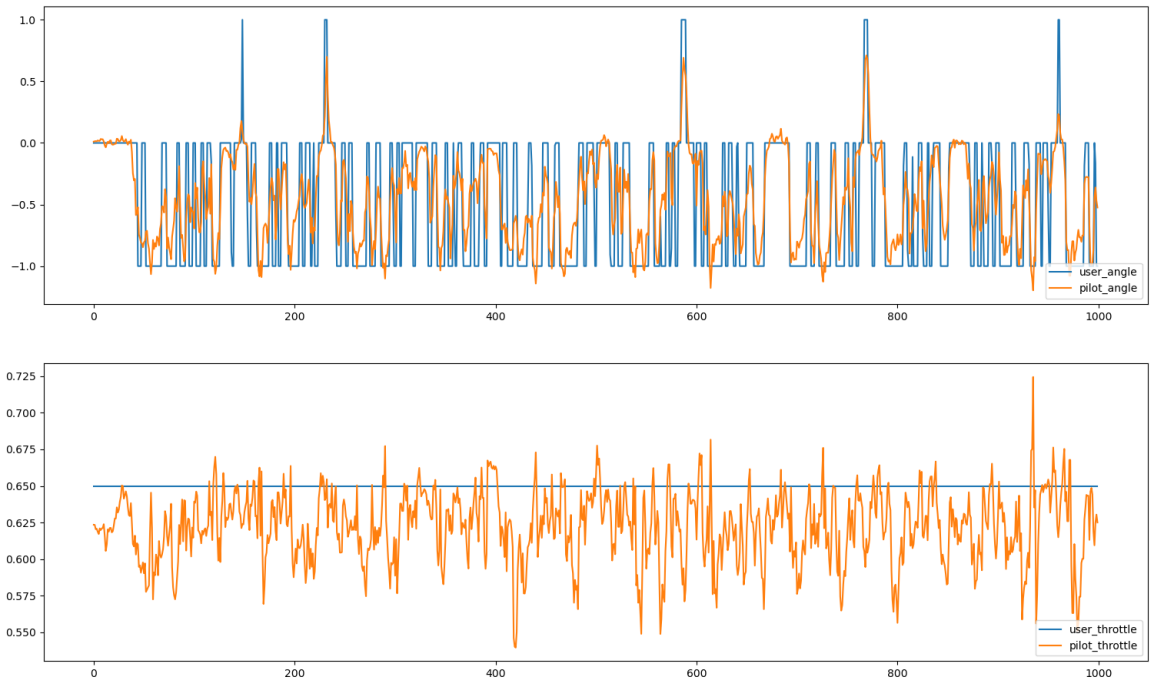
Finalment, en avaluar la conducció del model D s'obté un resultat molt diferent de l'expectat. L'error del model era suficientment baix com per suposar un bon funcionament del vehicle. Tanmateix, en posar-lo en marxa es descontrola a mesura que circula pel carril fins que, arribat un punt, perd la direcció i surt de la pista. En quant a la incorporació a pista, aconsegueix incorporar-se en alguns dels intents realitzats, però amb moltes dificultats.

A la secció 6.1.3 s'ha explicat la importància de la qualitat i la quantitat de les dades a l'hora d'entrenar un model. En el cas del model C, les dades d'entrada es divideixen en tres grups principals: gir a l'esquerra, gir a la dreta i camí recte. La quantitat de dades disponibles en cada grup resulta ser suficient per entrenar correctament el model. D'altra banda, en el cas del model D, a més a més dels tres grups anteriors s'afegeix la circulació per fora de pista. Això comporta una reducció de la quantitat de les dades disponibles en cada grup, ja que la memòria RAM de l'ordinador virtual no permet augmentar el nombre total de dades d'entrada. Per aquest motiu no s'ha aconseguit entrenar correctament el model i, per tant, no és vàlid.

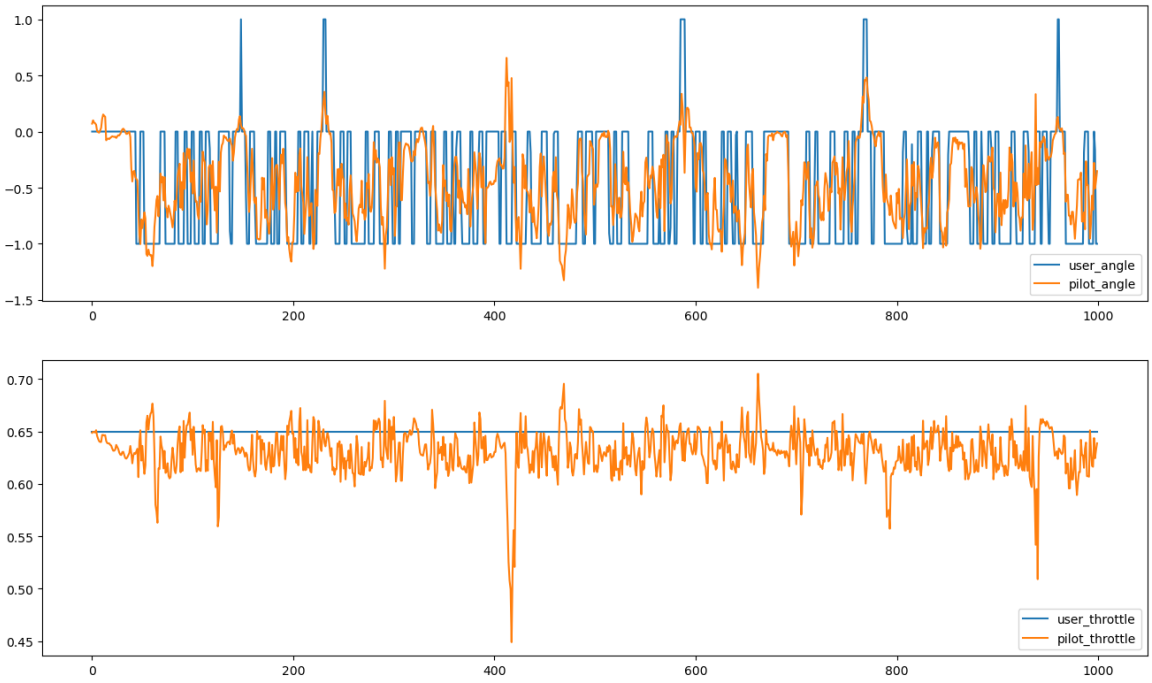
6.2.2 Prediccions dels models

El criteri de les prediccions dels models consisteix en observar la resposta d'un model entrenat davant una nova base de dades (imatges d'entrada). Tot i haver determinat la validesa dels models de conducció autònoma mitjançant la posada en marxa del vehicle, s'analitzaran dues prediccions realitzades, una del model C i l'altra del model D, davant una mateixa base de dades (en aquest cas, en sentit antihorari).

A la Figura 6.3a s'observa la predicció del model C davant les dades d'entrada. La gràfica de la part superior representa l'angle del model (en color taronja) respecte a l'angle de l'usuari que controla manualment el vehicle (en color blau). En aquest cas la resposta del model és precisa, assignant valors PWM molt propers als d'entrada i de manera fluïda. En quant a la gràfica inferior, equivalent a l'acceleració dels motors DC, existeix una petita desviació entre el valor de l'usuari (en color blau) i



(a) Predicció Model C.



(b) Predicció Model D.

Figura 6.3: Prediccions dels models davant una base de dades.

el del model (en color taronja). Com s'ha comentat prèviament a la secció 5.2.2, l'entrenament lineal pot errar en l'aprenentatge de l'acceleració. Tot i així, la diferència en l'acceleració no suposa un problema en la conducció autònoma del vehicle.

A la Figura 6.3b s'observa una predicció del model D bastant similar a l'anterior. Tanmateix, en aquest cas hi ha situacions en les quals el model s'equivoca i en comptes de realitzar un gir en un

sentit, el realitza en el sentit contrari. El fet d'haver estat entrenat amb imatges d'entrada a pista pot crear aquest tipus d'interferències. En quant a la predicció de l'acceleració del model, és pràcticament idèntica al primer model.

6.3 Resultats numèrics

A continuació es pot observar la Taula 6.1 amb el resum dels resultats numèrics obtinguts. Aquesta taula conté el número d'èpoques, el temps d'entrenament i l'error acumulat de cada model, així com la seva validesa com a model per la conducció autònoma.

Taula 6.1: Resum dels resultats numèrics obtinguts.

Model	Èpoques	Temps d'entrenament (min)	Error acumulat	Avaluació
A	17	7,16	0,1433	No vàlid
B	19	9,72	0,1481	No vàlid
C	18	12,90	0,1313	Vàlid
D	22	14,64	0,1333	No vàlid

L'error acumulat del model ens indica quant de correcte ha sigut l'entrenament del model. Tot i així, un correcte entrenament no necessàriament implica un model vàlid. Com s'ha observat en l'avaluació de l'experiment D (secció 6.2.1), un model ben entrenat amb un error baix ha resultat no ser vàlid per la conducció autònoma.

A partir de les premises i els resultats anteriors es pot afirmar que:

- L'error acumulat del model no depén de les èpoques o del temps d'entrenament.
- El model amb menor error acumulat resulta ser el model vàlid per la conducció autònoma del vehicle.
- La validesa del model depén de la qualitat i de la quantitat de les imatges utilitzades en l'entrenament, així com de l'error acumulat obtingut.

7 Resum del pressupost

El pressupost d'aquest projecte, *Deep Learning per la conducció autònoma de vehicles*, està format pels costos de material i els costos d'enginyeria. A la Taula 7.1 es pot observar el resum del pressupost total, sent aquest de 12232,17€. Per un desglossament dels costos més detallat, llegir la documentació del pressupost [42].

Table 7.1: Resum del pressupost total.

Pressupost	**Cost (€)**
Material	307,17
Enginyeria	11925
Total	**12232,17**

8 Conclusions i futures millores

8.1 Conclusions

En la realització d'aquest treball s'ha preparat i programat un vehicle de radio-control tipus Donkey-Car el qual, mitjançant la utilització de tècniques de Deep Learning, és capaç de circular per una pista tancada.

S'han complert els requeriments principals del projecte, implementant un model basat en una CNN capaç de generar una sortida per la direcció i una altra per l'acceleració del vehicle. A més a més, s'ha avaluat el comportament d'aquest a través de dos criteris diferents, confirmant així la seva validesa.

La principal dificultat del projecte ha estat la creació d'un model vàlid per la conducció autònoma a partir d'un ordinador virtual amb poca memòria RAM, ja que la quantitat d'imatges d'entrada de l'entrenament es troba limitada per la potència disponible; a més potència, més memòria d'entrada. Tanmateix, augmentant la memòria de l'ordinador i optimitzant la selecció de les dades d'entrada s'ha aconseguit solucionar el problema.

Quatre experiments han sigut efectuats, dels quals només un ha aconseguit desenvolupar un model capaç de moure's en un circuit tancat en ambdós sentits de la circulació, així com realitzar alguna maniobra d'entrada a pista. A partir dels resultats obtinguts en els experiments s'arriba a la conclusió que la validesa del model no depén únicament de l'error acumulat, sinó de la qualitat i quantitat de dades d'entrada emprades en l'entrenament. D'altra banda, la validesa del model és independent tant de les èpoques com del temps d'entrenament.

Finalment, concloure la secció destacant la importància del Deep Learning i la multitud d'aplicacions relacionades amb la vida quotidiana de les persones, com per exemple la detecció de càncers a partir de xarxes neuronals de tipus CNN, la classificació de productes alimentaris en el món de l'agricultura o la conducció autònoma de vehicles per tal de reduïr els accidents de trànsit.

8.2 Futures millores

En aquesta secció es presentaran les futures modificacions del treball per tal de millorar la conducció autònoma del vehicle. Es diviran en dos tipus, segons si són tasques a curt termini (CT) o a llarg termini (LT).

- **Entrenament en GPUs (CT).** L'entrenament dels models s'ha vist limitat per la potència del *hardware* de l'ordinador virtual. Traslladant l'algoritme d'entrenament a plataformes com per exemple *Google Colab*, on l'entorn d'execució és molt més potent, permetria utilitzar una base de dades major i crear un model capaç de circular tant per dins com per fora del circuit.
- **Circuit més complex (CT).** Per obtenir un model de conducció més realista, el circuit de proves hauria de ser modificat. En comptes de ser un circuit en forma de 0 es podrien afegir més corbes i augmentar la distància del recorregut.
- **Sensors d'obstacles (LT).** A més a més de la càmera, afegir un sensor LIDAR (en anglés, *Laser Imaging Detection and Ranging*) per detectar els obstacles que es trobin al circuit, com podria ser un vianant creuant un pas de zebra.

- **Detecció de senyals de trànsit (LT).** Una vegada programat el vehicle per detectar els senyals vials (carril del circuit), el següent pas seria crear i implementar una xarxa neuronal capaç de detectar senyals de trànsit verticals, com per exemple un senyal de *stop* o un semàfor.

Agraïments

Primer de tot, agraïr al director d'aquest projecte, Jose Antonio Soria Perez, per l'ajut, el temps i els coneixements proporcionats durant tots aquests mesos.

A més a més, m'agradaria donar reconeixement al meu pare, la meva mare i el meu germà pel seu suport constant durant tot el meu pas per la universitat.

Finalment, donar les gràcies als meus amics i companys de carrera per ajudar-me sempre que ho he necessitat i fer d'aquests quatre anys una experiència inoblidable.

Bibliografia

[1] Alexander Amini. *MIT Introduction to Deep Learning*. Febr. de 2021. URL: https://www.youtube.com/watch?v=5tvmMX8r_OM&ab_channel=AlexanderAmini.

[2] Mindy Support. *How Machine Learning in Automotive Makes Self-Driving Cars a Reality*. Febr. de 2020. URL: https://mindy-support.com/news-post/how-machine-learning-in-automotive-makes-self-driving-cars-a-reality/.

[3] Abhishek Gupta et al. "Deep learning for object detection and scene perception in self-driving cars: Survey, challenges, and open issues". A: (jul. de 2021), pàg. 1 - 4. URL: https://reader.elsevier.com/reader/sd/pii/S2590005621000059?token=ECB3BEDD145C69F99918CD3BBC38A5ED536B0E595AF0425E60AF5DC7D3900BF2566062413963BEA41157AD8D87284B61& originRegion=eu-west-1&originCreation=20210612162935.

[4] World Health Organization. *Road traffic injuries*. Juny de 2021. URL: https://www.who.int/news-room/fact-sheets/detail/road-traffic-injuries.

[5] Waveshare. *PiRacer, AI Racing Robot Powered by Raspberry Pi 4, Supports DonkeyCar Project*. 2021. URL: https://www.waveshare.com/piracer-ai-kit.htm.

[6] Ine Jernelv. *Convolutional neural networks for classification and regression analysis of one-dimensional spectral data*. Maig de 2020. URL: https://arxiv.org/abs/2005.07530.

[7] Waveshare. *Donkeycar: a python self driving library*. 2021. URL: https://github.com/waveshare/donkeycar.

[8] Maria Gorini. *What Is the Difference Between Machine Learning and Deep Learning?* Juny de 2019. URL: https://blog.bismart.com/en/difference-between-machine-learning-deep-learning.

[9] Oliver Theobald. "Machine Learning For Absolute Beginners". A: 2017, pàg. 15 - 21.

[10] Julianna Delua. *Supervised vs. Unsupervised Learning: What is the Difference?* Març de 2021. URL: https://www.ibm.com/cloud/blog/supervised-vs-unsupervised-learning.

[11] Guru99. *Supervised vs Unsupervised Learning: Key Differences*. Juny de 2021. URL: https://www.guru99.com/supervised-vs-unsupervised-learning.html.

[12] Devin Soni. *Supervised vs. Unsupervised Learning*. Març de 2018. URL: https://towardsdatascience.com/supervised-vs-unsupervised-learning-14f68e32ea8d.

[13] GateVidyalay. *K-Means Clustering Algorithm | Examples*. 2020. URL: https://www.gatevidyalay.com/k-means-clustering-algorithm-example/.

[14] Shweta Bhatt. *5 Things You Need to Know about Reinforcement Learning*. Març de 2018. URL: https://www.kdnuggets.com/2018/03/5-things-reinforcement-learning.html.

[15] Bootcamp AI. *Reinforcement Learning - Aprendizaje por refuerzo*. Març de 2020. URL: https://bootcampai.medium.com/reinforcement-learning-aprendizaje-por-refuerzo-c34bb085bb5.

[16] Ximena Islas. *Qué es un perceptrón? La red neuronal artificial más antigua*. Febr. de 2021. URL: https://www.crehana.com/es/blog/tech/que-es-perceptron-algoritmo/#que-es-perceptron.

[17] DeepAI. *Perceptron*. 2020. URL: https://deepai.org/machine-learning-glossary-and-terms/perceptron.

[18] DeepAI. *Hidden Layer*. 2020. URL: https://deepai.org/machine-learning-glossary-and-terms/hidden-layer-machine-learning.

[19] Simeon Kostadinov. *Understanding Backpropagation Algorithm*. Juny de 2019. URL: https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd.

[20] Shiv Vignesh. *The Perfect Fit for a DNN*. Jul. de 2020. URL: https://medium.com/analytics-vidhya/the-perfect-fit-for-a-dnn-596954c9ea39.

[21] DataRobot. *Underfitting*. 2018. URL: https://www.datarobot.com/wiki/underfitting/.

[22] Yann LeCun, Yoshua Bengio i Geoffrey Hinton. "Deep Learning". A: (maig de 2015). URL: https://www.researchgate.net/publication/277411157_Deep_Learning.

[23] Sumit Saha. *A Comprehensive Guide to Convolutional Neural Networks - the ELI5 way*. Des. de 2018. URL: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[24] Ismail Mebsout. *Convolutional Neural Networks' mathematics*. Febr. de 2020. URL: https://towardsdatascience.com/convolutional-neural-networks-mathematics-1beb3e6447c0.

[25] Ben Dickson. *What are convolutional neural networks (CNN)?* Gener de 2020. URL: https://bdtechtalks.com/2020/01/06/convolutional-neural-networks-cnn-convnets/.

[26] Juan Barrios. *Redes Neuronales Convolucionales*. URL: https://www.juanbarrios.com/redes-neurales-convolucionales/#Como_estan_construidas_y_como_funcionan.

[27] Ava Soleimany. *MIT Recurrent Neural Networks*. Febr. de 2021. URL: https://www.youtube.com/watch?v=qjrad0V0uJE&ab_channel=AlexanderAmini.

[28] Andrej Karpathy. *The Unreasonable Effectiveness of Recurrent Neural Networks*. 2015. URL: http://karpathy.github.io/2015/05/21/rnn-effectiveness/.

[29] IBM Cloud Education. *Recurrent Neural Networks*. Set. de 2020. URL: https://www.ibm.com/cloud/learn/recurrent-neural-networks.

[30] Adrián Hernández. *Problema del desvanecimiento del gradiente (vanishing gradient problem)*. Maig de 2018. URL: https://mlearninglab.com/2018/05/06/problema-de-desvanecimiento-del-gradiente-vanishing-gradient-problem/.

[31] Harshall Lamba. *Understanding Semantic Segmentation with UNET*. Febr. de 2019. URL: https://towardsdatascience.com/understanding-semantic-segmentation-with-unet-6be4f42d4b47.

[32] Waveshare. *PiRacer Assembly Manual*. Febr. de 2020. URL: https://www.waveshare.com/wiki/PiRacer_Assembly_Manual.

[33] Waveshare. *DonkeyCar for Pi-Setup Raspberry Pi*. Abril de 2021. URL: https://www.waveshare.com/wiki/DonkeyCar_for_Pi-Setup_Raspberry_Pi.

[34] Raspberry Pi. *Raspberry Pi OS*. 2021. URL: https://www.raspberrypi.org/software/.

[35] Raspberry Pi. *Raspberry Pi OS*. 2021. URL: https://www.raspberrypi.org/documentation/installation/installing-images/.

[36] DonkeyCar. *How to Build a Donkey*. 2020. URL: https://docs.donkeycar.com/guide/build_hardware/.

[37] Red Hat. *Red Hat Enterprise Linux 4*. 2020. URL: https://web.mit.edu/rhel-doc/4/RH-DOCS/rhel-rg-es-4/ch-ssh.html.

[38] Greenend. *Download PuTTY: latest release (0.75)*. Maig de 2021. URL: https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html.

[39] Luis Llamas. *Controlar 16 servos o 16 salidas PWM en Arduino con PCA9685*. Nov. de 2016. URL: https://www.luisllamas.es/controlar-16-servos-o-16-salidas-pwm-en-arduino-con-pca9685/.

[40] DonkeyCar. *What is a Part*. 2020. URL: https://docs.donkeycar.com/parts/about/.

[41] DonkeyCar. *Keras Parts*. 2020. URL: https://docs.donkeycar.com/parts/keras/.

[42] Carles Montilla. "Deep Learning per la conducció autònoma de vehicles - Pressupost". A: (juny de 2021), pàg. 1 - 2.