TecnoCampus
Escola Superior Politècnica

Centre adscrit a la
upf. Universitat Pompeu Fabra Barcelona

Enginyeria Tècnica Industrial: Especialitat Electrònica Industrial

MÈTODE SELECTIU DE CAPTURA D'OCELLS PER A LA SEVA CONSERVACIÓ

Memòria

ESTER PEINADO
PONENT: XAVIER FONT

PRIMAVERA 2020

TecnoCampus
Mataró-Maresme

Agraïments

A l'Arnau, per escoltar-me, guiar-me i recolzar-me en tot moment.

A l'Antoni i la Isabel, per creure en mi i ajudar-me a arribar on sóc ara.

A tots els companys i companyes de Rockwell Automation, per les seves esbojarrades idees i per contagiar-me la seva alegria i energia cada dia.

Resum

El present projecte tracta del desenvolupament d'un mètode selectiu de captura d'ocells per tal d'ajudar a l'organització SEO/BirdLife en la protecció i conservació de les aus.

La seva realització parteix del disseny i fabricació d'una trampa de fusta a la qual se li incorporen dispositius electrònics. També s'inclou el desenvolupament, mitjançant xarxes neuronals, d'un sistema de reconeixement d'imatges capaç d'identificar quatre espècies diferents d'aus.

Resumen

El presente proyecto trata del desarrollo de un método selectivo de captura de pájaros para ayudar a la organización SEO/BirdLife en la protección y conservación de las aves.

Su realización parte del diseño y fabricación de una trampa de madera a la cual se le incorporan dispositivos electrónicos. También se incluye el desarrollo, mediante redes neuronales, de un sistema de reconocimiento de imágenes capaz de identificar cuatro especies diferentes de aves.

Abstract

The present project deals with the development of a selective bird-capture method in order to help SEO/BirdLife organization with the bird's protection and conservation.

Its realization is based on the design and manufacturing of a wooden trap with electronic devices incorporated. It also includes the development, using neural networks, of an image recognition system capable of identifying four different bird species.

Índex.

Índex de figures

Índex de taules

Glossari de termes

API	Interfície de Programació d'Aplicacions (*Application Programming Interface*)
b	Biaix (*bias*)
CNN	Xarxa Neuronal Convolucional (*Convolutional Neural Network*)
f	Dimensió del filtre
FSR	Sensor de Força Resistiu (*Force-Sensing Resistor*)
g	Funció d'activació no lineal
GUI	Interfície Gràfica d'Usuari (*Graphic User Interface*)
IA	Intel·ligència Artificial
J	Funció de pèrdua
MIT	Institut de Tecnologia de Massachusetts (*Massachusetts Institute of Technology*)
n	Dimensió de la imatge
OOM	Error produït per manca de memòria (*Out Of Memory*)
p	Padding en una xarxa neuronal
RPi	Placa de desenvolupament Raspberry Pi
SEO	Societat Espanyola d'Ornitologia
SIOC	Servidor d'Informació Ornitològica de Catalunya
s	Stride en una xarxa neuronal
TPU	Unitat de Processament Tensorial (*Tensor Processing Unit*)
UICN	Unió Internacional per a la Conservació de la Natura

VNC	Computació Virtual de Xarxa (*Virtual Network Computing*)
w	Pes associat a les entrades en una xarxa neuronal (*weight*)
w^T	Vector dels pesos associat a les entrades en una xarxa neuronal transposat
X	Vector de característiques d'entrada de la xarxa neuronal
$\hat{y}$	Sortida de la xarxa neuronal, valor predit
y	Sortida esperada de la xarxa neuronal, valor real
$\mathcal{L}$	Funció de cost

1. Objectius.

1.1. Propòsit.

Proporcionar als ornitòlegs de l'organització SEO/BirdLife un mètode de captura selectiu.

1.2. Finalitat.

Possibilitar l'estudi concret d'una única espècie alhora que s'optimitza el temps d'utilització de les trampes, permetent a l'organització SEO/BirdLife destriar i agilitzar les captures d'ocells per al seu posterior marcatge i control.

1.3. Objecte.

Disseny i prototipatge d'una trampa de caràcter portable la qual estarà dotada d'un sistema de detecció d'objectes i de connexió a internet per tal d'enviar un senyal a la central quan la trampa estigui activada. Una interfície d'usuari des de la qual decidir quina au es vol estudiar.

1.4. Abast.

El sistema de detecció d'objectes serà capaç d'identificar quatre espècies diferents d'aus: cadernera, pardal comú, passerell comú i pinsà borroner. S'utilitzarà un controlador amb connexió WiFi i s'incorporarà un mòdem USB que, juntament amb una targeta SIM, actuarà com a punt de connexió a internet per al controlador.

Quant al disseny funcional, el sistema serà portable. Es realitzarà una trampa en forma de casa d'ocells a l'interior de la qual es dipositarà l'esquer. Disposarà d'una porta amb desplaçament vertical aguantada per un actuador que la deixarà caure quan es realitzi una identificació positiva de l'animal del seu interior. Per a garantir la seva seguretat, s'incorporarà un detector de presència.

No s'inclou el disseny del sistema productiu i logístic associat, doncs l'objectiu és proporcionar una solució, no un producte comercial. Tampoc s'inclouen els estudis i proves necessàries per a l'homologació del prototip.

2. Antecedents i necessitats d'informació.

BirdLife International és una associació mundial d'organitzacions de conservació (ONG) que s'esforça per conservar les aus, els seus hàbitats i la biodiversitat global. Actualment hi ha 121 BirdLife Partners a tot el món, un per país o territori [1].

SEO/BirdLife és l'organització representant de BirdLife International a Espanya, la qual compta amb 31.000 voluntaris, 11 oficines i 43 grups locals. Entre els projectes realitzats per a conservar les aus i els seus hàbitats estan la realització de censos i el seguiment de poblacions.

La iniciativa Paser és l'encarregada de realitzar el seguiment de poblacions d'aus mitjançant la seva captura i marcatge amb anelles. Tal com explica la mateixa organització: "Això facilita informació que és difícil d'obtenir mitjançant el seguiment de poblacions a través de censos o simple observació i és especialment important per conèixer taxes de supervivència d'espècies, longevitats, com migren o aspectes de la seva fenologia, entre moltes altres qüestions" [2].

En certes ocasions és necessari realitzar un estudi en profunditat d'una espècie en concret. Per tal de fer més eficient el procés de captura d'aus en aquests casos, SEO/BirdLife vol iniciar un projecte per dur a terme captures intel·ligents. Després d'una reunió amb Jordi Prieto [3], membre de la delegació SEO/BirdLife a Catalunya, s'estableix que es realitzarà un primer estudi d'aus voladores residents a Catalunya de longitud igual o inferior als 15 cm. Per a capturar la majoria d'aquests ocells, es pot emprar una mateixa trampa variant l'esquer del seu interior. La trampa tindrà una porta, la qual romandrà oberta fins que es detecti presència al seu interior, moment en el qual la càmera es posarà en funcionament per a identificar si l'ocell pertany a l'espècie que es vol estudiar. En cas de resultar un reconeixement positiu, la porta es tancarà fins que una persona l'obri manualment.

L'objectiu és dissenyar un nou mètode de captura, alhora que es dota les trampes d'un cert nivell d'intel·ligència. D'aquesta manera, s'espera poder ser més selectius amb les captures d'aquests animals. A més a més, es preveu que aquest mètode suposi una reducció del temps que la trampa està bloquejada, evitant la captura d'un ocell no desitjat i enviant un avís a la central un cop tancada la porta. Això suposa una optimització del seu temps d'ús.

Un cop implementat el projecte, els resultats obtinguts es presentaran a BirdLife International per tal d'expandir-lo a la resta d'organitzacions membres.

2.1. Normativa.

2.1.1. Catalunya.

- Decret 148/1992, de 9 de juny, pel qual es regulen les activitats fotogràfiques, científiques i esportives que poden afectar les espècies de la fauna salvatge.
- Decret Legislatiu 2/2008, de 15 d'abril, pel qual s'aprova el Text refós de la Llei de protecció dels animals.

2.1.2. Estat espanyol.

- Llei 4/1989, de 27 de març, de conservació dels espais naturals i de la flora i fauna silvestres.
- Llei 42/2007, de 13 de desembre, del patrimoni natural i de la biodiversitat.
- Reial Decret 139/2011, de 4 de febrer, per al desenvolupament del Llistat d'Espècies Silvestres en Règim de Protecció Especial i del Catàleg Espanyol d'Espècies Amenaçades.

2.1.3. Internacional i Unió Europea.

- Directiva 92/43/CEE del Consell, de 21 de maig, relativa a la conservació dels hàbitats naturals i de la fauna i la flora silvestres.
- Directiva 97/62/CE del Consell, de 17 d'octubre, per la qual s'adapta al progrés científic i tècnic la Directiva 92/43/CEE.
- Directiva 2009/147/CE del Parlament Europeu i del Consell, de 30 de novembre, relativa a la conservació dels ocells silvestres.

2.2. Aplicacions de reconeixement d'ocells existents.

Actualment s'han creat diverses aplicacions orientades a la identificació d'ocells. Tot i que cap d'elles cobreix per complet les necessitats plantejades, serveixen d'inspiració per al desenvolupament de la solució. El principal problema resideix en el fet que no són aplicacions de reconeixement en temps real.

2.2.1. eBird.

Aplicació per a mòbils desenvolupada per eBird, una base de dades d'observacions d'ocells mundial. S'ha desenvolupat en col·laboració amb la universitat de Cornell i permet registrar les aus que s'identifiquen, així com una àmplia informació sobre el lloc i les condicions de la seva observació [4].

2.2.2. Merlin Bird ID.

Aplicació creada per la universitat de Cornell que permet a l'usuari identificar l'espècie que ha observat. Responent tres preguntes senzilles sobre l'au que es vol reconèixer o carregant una fotografia, aquesta app ofereix una llista de possibles coincidències [5], [6].

2.2.3. Aves de España.

Aquesta ha sigut realitzada per SEO/BirdLife. Igual que en el cas anterior, l'usuari respon a unes preguntes que genera l'aplicació. En aquest cas, a mesura que es responen les preguntes automàticament es van reduint les solucions mostrades, fins que finalment resta una única solució [7], [8].

2.3. Bases de dades.

Per a desenvolupar una xarxa neuronal capaç de reconèixer objectes en imatges, és necessari entrenar-la prèviament. Les xarxes neuronals s'entrenen a partir de bases de dades, en aquest cas una base d'imatges, a partir de les quals identifiquen patrons i generen regles (vegeu apartat 2.5).

S'ha realitzat una cerca de bases d'imatges d'ocells i se n'han trobat diverses com eBird o Avibase entre d'altres. Les imatges d'aquests portals web són accessibles però no es poden descarregar, la qual cosa planteja un problema.

Després de contactar amb la plataforma sol·licitant accés a la base de dades sense obtenir cap resposta, s'ha decidit que aquesta es crearà personalment.

2.4. Mètodes de captura.

Actualment, existeix una gran varietat de mètodes de captura, tot i que molts d'ells no són legals. A continuació es presenten els mètodes més utilitzats [3], [9].

2.4.1. Xarxa abatible.

En aquest cas es lliga una xarxa a una estructura de pals. A més a més, es col·loca un fil de niló, imperceptible per a l'ocell, a certa alçada entre els pals. Quan l'au vola, estira del fil i els pals cauen. S'ha de tenir present que, tot i ser legal, el mètode suposa un risc per a l'animal pel fet que el pal pot caure-li a sobre i ferir-lo.

Aquest tipus de trampa es col·loca principalment en nius, ja que és un lloc per on s'assegura que l'ocell volarà i a més permet triar l'espècie que es vol capturar. S'empra majoritàriament per a ocells de mida mitjana, com per exemple la cigonya blanca. Tot i això, no és un mètode efectiu al 100%, ja que pot donar-se el cas que la xarxa no caigui sobre l'ocell.

2.4.2. Xarxa japonesa.

És el mètode més emprat pels ornitòlegs de SEO/BirdLife. Consisteix en una xarxa formada per fils molt fins i generalment de color negre que es col·loca entre dos pals de manera que queda vertical. Aquesta es divideix en diferents franges horitzontals, cadascuna de les quals forma una bossa que penja per un dels costats tal com mostra la Fig. 2.1. La mida dels forats de la malla i el nombre de franges varia en funció de les espècies d'ocells que es vulguin capturar.

La trampa es col·loca de matinada, quan encara és fosc, de manera que quan les aus es desperten i comencen a volar no les veuen i s'hi queden enredades.

Aquesta pràctica només està permesa amb fins científics i per tant es requereixen unes llicències i permisos per a poder realitzar-la.

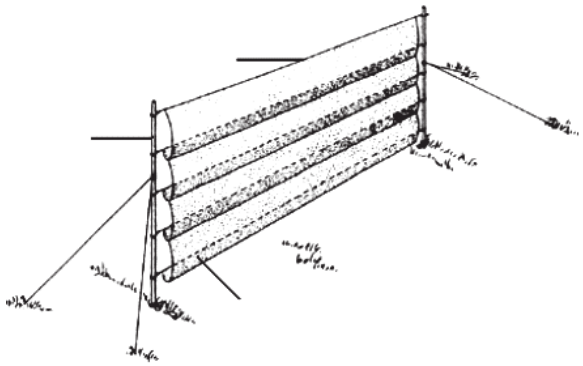


Fig. 2.1. Xarxa japonesa.
Font: www.researchgate.net

2.4.3. Caça amb vesc.

És un mètode emprat per a caçar ocells cantaires. Consisteix a col·locar palets en forma de creu (anomenats creueres) i untar-los amb vesc. Aquests es disposen entre les branques d'un arbret i s'amaga un ocell engabiat prop de l'arbre.

D'aquesta manera, quan l'ocell engabiat canta atrau d'altres de la seva mateixa espècie, els quals en posar-se sobre les branques es queden enganxats per culpa del vesc i no poden volar.

És un mètode de caça il·legal. Està prohibit ja que provoca la pèrdua de plomes tant de les ales com de la cua, resultant en la pèrdua total o parcial de la capacitat de vol de molts ocells. Desgraciadament se segueix practicant per caçar aus de cant i participar en concursos.

2.4.4. Comparació de mètodes.

A continuació es mostra una comparació dels diferents mètodes de captura explicats i la solució selectiva que es vol dissenyar.

Mètode	**Xarxa abatible**	**Xarxa japonesa**	**Caça amb vesc**	**Solució selectiva**
Permès	Sí	Sí	No	Sí
Selectiu	Sí	No	Sí	Sí
Eficaç	No	Sí	Sí	Sí

Taula 2.1. Comparació dels mètodes de captura.

2.5. Detecció d'objectes.

2.5.1. Intel·ligència artificial.

La intel·ligència artificial (IA) és la combinació d'algoritmes plantejats amb el propòsit de dotar màquines de les mateixes capacitats cognitives que té l'ésser humà, incloent-hi l'aprenentatge, el raonament i l'autocorrecció entre d'altres.

Per arribar a ser capaç d'emular el cervell humà, primer se l'ha d'ensenyar. Si per exemple, es vol emprar per a identificar imatges d'ocells, com és el cas, primerament ha de processar centenars d'imatges d'ocells per aprendre a distingir-los.

Seguidament es comença l'entrenament: rep imatges de diferents animals i ha de destriar les que són d'aus. Al principi falla i se li ha d'indicar quines imatges ha encertat i en quines s'ha equivocat. La IA aprendrà perquè falla i millorarà.

Finalment arriba un punt en què és capaç de treballar sola, sense rebre ordres. Simplement se li entreguen les dades d'entrada (imatges) i ella genera resultats (imatges d'ocells) sense que existeixi un codi que li indiqui els passos a seguir.

Aquesta estructura d'aprenentatge, entrenament i resultats és comuna per a les IA que realitzen tasques mecàniques i repetitives [10].

Es defineixen quatre tipus diferents d'intel·ligència artificial:

- Sistemes que pensen com humans: automatitzen activitats com la presa de decisions, la resolució de problemes i l'aprenentatge (p. ex.: les xarxes neuronals).
- Sistemes que actuen com humans: realitzen tasques de forma similar a com ho fan les persones (p. ex.: els robots).
- Sistemes que pensen racionalment: intenten emular el pensament lògic racional dels humans, podent percebre, raonar i actuar en conseqüència (p. ex.: els sistemes experts).
- Sistemes que actuen racionalment: intenten imitar de manera racional el comportament humà (p. ex.: els agents intel·ligents).

El present projecte se centrarà en xarxes neuronals per a la detecció d'objectes.

2.5.2. Machine learning i deep learning.

Abans d'explicar les xarxes neuronals és necessari introduir dos conceptes nous: el *machine learning* i el *deep learning*.

Tal com explica Alexander Amini, professor del MIT, el *machine learning* o aprenentatge automàtic és un subconjunt de la IA que s'enfoca en ensenyar a un algorisme com obtenir informació de les dades sense la necessitat de ser programat explícitament. Existeixen dos tipus: supervisat, on una persona li indica què fa bé i què no, i el no supervisat, on és la mateixa IA la que ha d'aprendre a saber què fa bé i què no a partir d'uns patrons.

L'aprenentatge profund o *deep learning* va un pas més enllà, extraient aquests patrons automàticament de les dades sense processar i eliminant doncs la necessitat que les regles siguin programades prèviament. Normalment s'implementa utilitzant una xarxa neuronal. El terme "profund" fa referència al nombre de capes ocultes de la xarxa (com més capes ocultes, més profunda és la xarxa). Les xarxes neuronals tradicionals utilitzen únicament 2 o 3 capes, mentre que les xarxes profundes en tenen centenars.

2.5.3. Transfer learning.

Els models d'aprenentatge profund sofisticats tenen milions de paràmetres i entrenar-los des de zero sovint requereix grans quantitats de dades de recursos informàtics. El *transfer learning* és una tècnica que redueix aquests requisits prenent un model que ja s'ha entrenat en una tasca relacionada i reutilitzant-lo en un nou model.

Per exemple, per crear un sistema de detecció d'objectes es pot aprofitar un model que ja estigui entrenat per reconèixer milers d'objectes diferents dins d'imatges. Es poden adaptar els coneixements existents del model pre-entrenat per detectar les classes d'imatges desitjades mitjançant un volum de dades de formació molt menor al que ha requerit el model original.

És útil per desenvolupar ràpidament nous models, així com per personalitzar models en entorns amb restriccions de recursos.

3. Objectius i especificacions tècniques.

A continuació s'exposen els objectius del projecte amb les especificacions tècniques associades:

- Identificar l'espècie de l'au que està a l'interior de la trampa.
  - Raspberry Pi 3 Model B.
  - Càmera Module V2.
  - Plataforma de codi obert per a realitzar xarxes neuronals.
- Dispositiu portàtil.
  - Bateria portàtil. Tensió de sortida 5 V DC. Corrent de sortida 2 – 2,5 A.
  - Cable Micro USB.
- Capturar ocells.
  - Trampa de fusta amb forats per augmentar la lluminositat.
  - Sensor FSR.
  - Porta de fusta amb una guia.
  - Solenoide de 5 V.
- Saber si la trampa s'ha activat des de qualsevol punt.
  - Comunicació bidireccional entre la central i la trampa.
  - Mòdem USB.

4. Disseny de la trampa.

El disseny de la trampa s'ha inspirat en una casa niu per a ocells. S'han modificat les mesures i alguns elements com les parets laterals i frontal per dotar-la de les funcionalitats necessàries.

Fig. 4.1. Model 3D de la trampa.

Primerament, s'ha canviat l'entrada de la part frontal. En comptes de ser un orifici rodó i de diàmetre petit s'ha decidit fer una entrada rectangular de 8 x 9 cm. A sobre d'aquesta entrada hi ha una porta enreixada la qual s'aguanta per un actuador i, quan aquest es retreu, la porta es desplaça verticalment fins a tapar el forat. Als laterals de l'entrada se li han afegit unes guies per on llisca la porta.

Les dues parets laterals compten amb una finestra enreixada de 13 x 16 cm, les quals augmenten la lluminositat i la ventilació de l'habitacle, millorant així l'entorn per a l'animal alhora que es milloren les condicions de llum per a la correcta captació d'imatges. La finestra esquerra tindrà una frontissa per poder obrir-la i facilitar l'extracció de l'ocell.

D'altra banda, la peça inferior s'ha allargat uns mil·límetres perquè faci de topall quan baixa la porta. A més a més, s'ha afegit una altra peça de les mateixes dimensions que la inferior sense el topall a 15 cm d'alçada. En aquest suport serà on es col·locaran tots els dispositius electrònics necessaris per a la realització del reconeixement de l'au, com per exemple la placa i una bateria. Aquesta plataforma compta amb diversos forats de 5 mm de diàmetre per afavorir la ventilació dels dispositius electrònics.

La forma inclinada del sostre està pensada per facilitar que l'aigua llisqui en cas de pluja i sobresurt tant pels laterals com per la part fontal per tal d'evitar que l'aigua caigui directament sobre les parets.

Per últim, s'ha augmentat la profunditat de l'habitacle. L'objectiu és augmentar la distància entre l'esquer, el qual es col·locarà al fons, i la porta per tal que doni temps a tancar-la de manera segura quan es realitzi un reconeixement positiu de l'au.

Els plànols de la trampa es poden trobar al document corresponent. En aquests esquemes es pot contemplar que totes les peces tenen un perfil emmerletat. Se li ha donat aquesta forma per tal de facilitar el posterior muntatge i, sobretot, per tal d'augmentar-ne la robustesa.

4.1. Realització de la trampa.

Un cop dissenyada l'estructura, toca construir-la. Primerament, es contacta amb la universitat per llogar la talladora làser. Deguda la situació excepcional que s'està vivint, s'acorda enviar els arxius i passar a buscar les peces un cop tallades, de manera que els encarregats del laboratori s'ocupen de la màquina. Es converteixen les peces dissenyades amb SolidWorks a un format vectorial, DWG, i s'envien al laboratori per ser tallades en fusta de balsa de 5 mm. En tallar la fusta amb el làser, aquesta queda cremada. Per evitar que les parts ennegrides taquin en tocar-les, es procedeix a llimar tots els costats per eliminar les cendres.

Per tal que l'actuador aguanti la porta, el pistó ha de passar a través de la guia. Aquesta està formada per dues peces, una d'elles ja té el forat fet però a la segona se li ha de donar forma amb una llima.

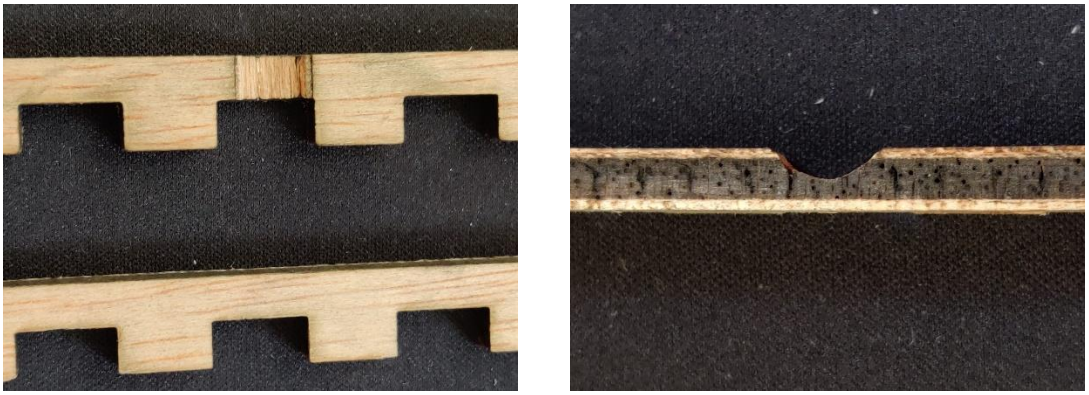


Fig. 4.2. Preparació de la guia perquè pugui passar el pistó.

Amb les peces a punt, es procedeix a encolar-les amb cola blanca i s'envernissa l'estructura. Finalment, es caragola la tanca i les frontisses del sostre i el lateral esquerre. La Fig. 4.3 mostra el resultat del muntatge de la trampa.

Fig. 4.3. Resultat del muntatge de la trampa.

5. Dispositius electrònics.

Com s'ha mencionat anteriorment, la porta resta aguantada per un actuador, el qual en activar-se la deixa caure i la trampa queda tancada.

A l'entrada de la trampa es col·loca un sensor FSR, el qual serveix per detectar la presència d'un ocell. Per tal d'estalviar bateria, la càmera resta apagada fins al moment que es detecta que un ocell ha entrat a la trampa. Aquest sensor també serveix per evitar que el tancament de la porta pugui ferir algun animal, ja que l'actuador no s'activarà a no ser que el sensor no estigui detectant cap presència.

5.1. Actuador.

S'empra un solenoide com a actuador. Aquest necessita una tensió de 5 V i un corrent d'1,1 A per a activar-se. Els pins GPIO de la placa poden subministrar com a màxim 50 mA. Per evitar causar danys a la placa, l'actuador s'alimenta mitjançant un MOSFET tipus N tal com mostra la Fig. 5.1.

La funció d'un microcontrolador és lliurar senyals de control, és a dir, tensions. Un dels avantatges que presenta el MOSFET enfront d'altres transistors és que es tracta d'un dispositiu commutat per voltatge. Aquest element s'acostuma a muntar juntament amb una resistència de *pull-down* entre la porta i el sortidor, la funció de la qual consisteix a assegurar un nivell lògic baix a la porta del transistor quan el pin GPIO no està activat. El seu valor ha de ser relativament alt perquè el corrent que hi circula a través sigui mínim.

La font de 5 V de la Raspberry Pi (d'ara en endavant RPi) pot subministrar fins a 1,5 A, suficients per a fer funcionar el solenoide. Com que no estarà constantment alimentat, sinó que només rebrà alimentació de manera esporàdica i durant un període de temps curt, no suposa un risc per a la placa.

Finalment s'implementa un díode entre els terminals de l'actuador per permetre que es descarregui i protegir el transistor.

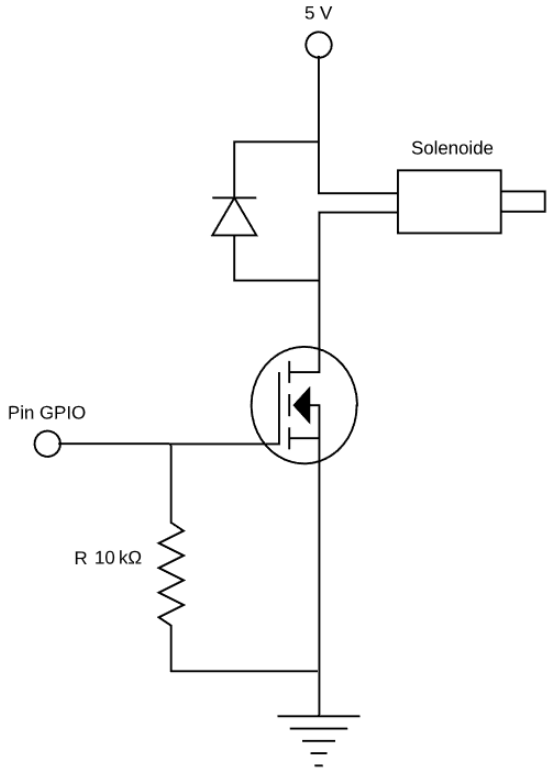


Fig. 5.1. Circuit de l'actuador.

5.2. Sensor FSR.

El sensor FSR varia la seva resistència en funció de la pressió que se li aplica. Sense pressió, el valor de la resistència és pròxima als 100 kΩ.

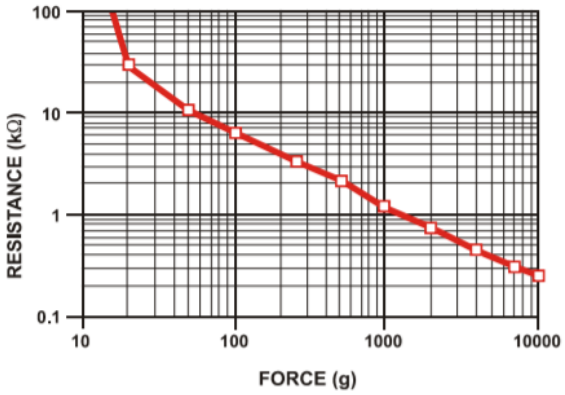
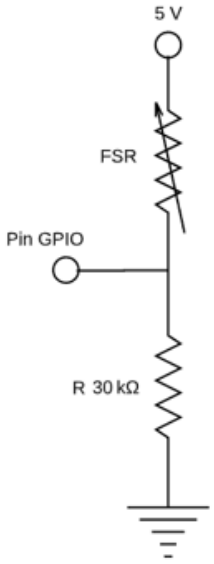


Fig. 5.2. Variació del valor de residència en funció de la força aplicada al sensor FSR.
Font: Datasheet – Interlink Electronics

Els ocells que entraran a la caseta mesuren com a màxim 15 cm de longitud i pesen entre 20 i 30 g. A partir de la Fig. 5.2 s'extreu que la resistència del sensor serà de 30 kΩ per a exemplars de 20 g i de 19 kΩ per a exemplars de 30 g. De les característiques de la placa se sap que els valors de *threshold* dels pins GPIO són:

- low <= 0,8 V
- high >= 2 V.

Per a activar l'entrada del pin en funció de la presència o absència d'un ocell, es dissenya un circuit divisor de tensió.



$$V_{GPIO} = 5 * \frac{R}{FSR + R} \geq 2\ V$$

FSR = 30 kΩ → V_{GPIO} = 2,5 V

FSR = 19 kΩ → V_{GPIO} = 3 V

Fig. 5.3. Circuit i càlculs dels valors del sensor FSR.

5.3. Mòdem USB.

És necessari poder establir una comunicació amb la RPi per tal de saber, per exemple, quan s'ha realitzat una captura. Una manera de fer-ho és mitjançant el software VNC. Aquesta aplicació permet que un client agafi el control d'un ordinador servidor remotament. Un cop instal·lat, és possible accedir al servidor des de qualsevol part del món i des de qualsevol dispositiu (ordinador, tauleta o mòbil) a través d'internet. A més a més, poden connectar-se simultàniament diversos clients a un mateix servidor, de tal manera que pot connectar-se l'ornitòleg que vol realitzar la captura amb el seu mòbil i alhora mantenir un control des de l'ordinador de la central.

L'aplicació de captura d'ocells s'implementarà, probablement, en entorns naturals amb connexió a internet limitada o nul·la, com poden ser la muntanya o un bosc. El mòdem USB és un router amb connexió a internet mòbil 4G o 3G, és a dir, amb una targeta SIM. Aquest dispositiu crea una xarxa WiFi 4G com a punt d'accés a la qual es connecta la RPi per poder establir la connexió VNC.

5.4. Muntatge.

Un cop dissenyats els circuits electrònics dels diferents dispositius, es procedeix a implementar-los a la trampa.

Fig. 5.4. Implementació de l'electrònica a la trampa.

6. Sistema de detecció d'objectes.

6.1. Xarxes neuronals.

Per facilitar la comprensió del desenvolupament posterior de dues xarxes neuronals, així com de l'aplicació de la tècnica *transfer learning*, s'ha considerat convenient explicar el funcionament i l'estructura de les xarxes neuronals [11].

6.1.1. Estructura d'una xarxa neuronal.

Per implementar les tècniques de *deep learning* s'utilitzen xarxes neuronals, les quals intenten emular el funcionament de les neurones dels éssers humans. Un perceptró és la unitat de la xarxa neuronal equivalent a una neurona. Per crear un perceptró primerament es defineixen unes entrades (x_1, x_2, ... , x_m), a cadascuna de les quals se li assigna un determinat pes (w_1, w_2, ... ,w_m). A continuació s'aplica una regla de propagació, la qual acostuma a ser una suma de les entrades ponderades amb els seus pesos corresponents, i un biaix (b, en anglès *bias*). Per acabar, es passa aquesta suma per una funció d'activació no lineal (g) amb la qual obtenim una sortida ($\hat{y}$). L'equació d'un perceptró es mostra a (6.1).

$$\hat{y} = g\left(b + \sum_{i=1}^{m} x_i\, w_i\right) \tag{6.1}$$

Cada pes w_i representa la influència relativa de l'entrada per la qual es multiplica, x_i. El biaix controla que tan predisposada està la neurona a generar un 1 o un 0 independentment dels pesos. Un biaix elevat fa que la neurona requereixi una entrada més alta per a generar una sortida d'1.

L'agrupació de neurones s'anomena capa. En una capa totes les neurones utilitzen les mateixes entrades però amb pesos associats diferents. També es poden combinar capes, de manera que la sortida d'un perceptró passa a ser l'entrada del següent. Combinant les diferents capes es forma una xarxa neuronal com la que es mostra a la Fig. 6.1.

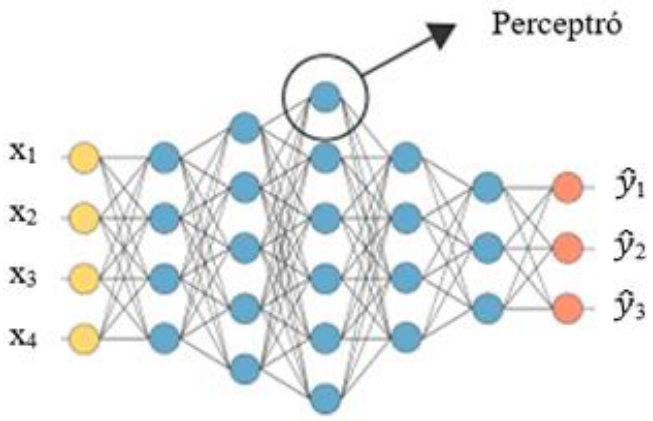


Fig. 6.1. Xarxa neuronal.
Font: www.introtodeeplearning.com

La capa d'entrada, formada pel vector de característiques d'entrada *X*, és la que rep la informació però no la processa, simplement actua com a receptora i emissora. Les capes intermèdies, anomenades capes ocultes, són aquelles que tant l'entrada com la sortida estan connectades amb una altra capa. La capa de sortida és la que genera els resultats de la xarxa, el valor de predicció $\hat{y}$.

Per arribar a reconèixer imatges, prèviament s'ha d'haver entrenat la xarxa. Es necessita un grup d'imatges les quals s'etiqueten segons l'objecte que contenen i que es vol reconèixer. En aquest punt, cada imatge (x_i) té associada una etiqueta (y_i). A partir d'aquestes dades, la xarxa comença a entendre les característiques concretes de cada etiqueta i les associa a la categoria corresponent. Finalment entrega una sortida ($\hat{y}_i$) amb un percentatge associat a cada categoria, indicant així el grau de pertinença de la fotografia a cada conjunt [12].

Durant el procés d'entrenament, s'ajusten els valors dels pesos i del *bias* per tal que el model realitzi una tasca concreta. Inicialment es parteix d'un conjunt de pesos aleatoris els quals es van redefinint iterativament fins a obtenir un resultat considerat com a bo. Hi ha tres tipus d'entrenament:

- Supervisat: s'indica un conjunt de patrons d'entrada juntament amb la sortida esperada. Els pesos s'ajusten de manera proporcional a l'error entre la sortida real i l'esperada. Aquest serà el mètode emprat en el present projecte.

- No supervisat: només s'indica un conjunt de patrons d'entrada, però no la sortida esperada. Els pesos s'ajusten en funció de la correlació existent entre les dades d'entrada.
- Per reforç: en aquest cas s'indica el conjunt de patrons d'entrada i se li diu a la xarxa si la sortida generada és o no correcta, però no se li dóna el valor de sortida esperat.

6.1.2. Generació del vector de característiques d'entrada *X*.

Quan es vol realitzar un problema d'etiquetatge de fotos, en aquest cas d'espècies d'ocells, l'entrada és una imatge i com a sortida es vol una etiqueta per tal de reconèixer si la imatge conté o no un ocell i, en cas afirmatiu, saber a quina espècie concreta pertany. L'objectiu és fer un classificador en el qual es pugui introduir una imatge representada per un vector d'entrada X i predir l'etiqueta corresponent.

En un ordinador, quan es guarda una imatge en realitat s'estan guardant tres matrius diferents corresponents als tres canals de color (vermell, verd i blau) de la imatge. Aleshores, si aquesta conté 64 x 64 píxels, realment es tenen tres matrius de dimensions 64 x 64 corresponents als valors de les intensitats dels píxels vermells, verds i blaus de la foto. Per convertir aquests valors de les intensitats dels píxels en un vector de característiques d'entrada X corresponent a la imatge, el que es fa és llistar primerament tots els píxels vermells, després tots els píxels verds i per últim tots els píxels blaus, obtenint un vector de característiques amb tots els valors d'intensitats de les tres matrius. Així doncs, si la imatge és de 64 x 64 píxels, la dimensió total del vector X serà de 64 x 64 x 3, és a dir, 12.288. S'utilitzarà la notació n_x = 12.288 per a representar la dimensió de les característiques d'entrada X.

Prenent com a exemple la Fig. 6.2, el vector d'entrada X seria el representat a (6.2).

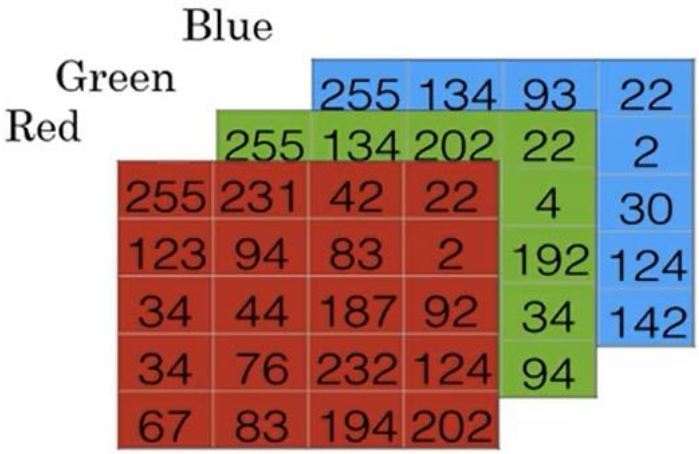


Fig. 6.2. Matrius dels tres canals de color d'una imatge.

Font: www.deeplearning.ai

$$X = \begin{bmatrix} 255 \\ 231 \\ . \\ . \\ . \\ 255 \\ 134 \\ . \\ . \\ . \\ 255 \\ 134 \\ . \\ . \\ . \\ 142 \end{bmatrix} \tag{6.2}$$

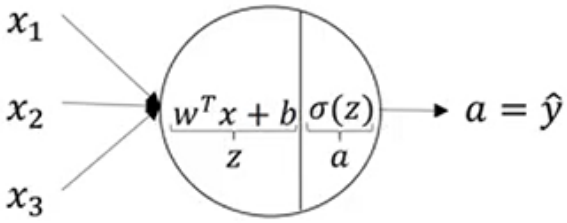
6.1.3. Obtenció de la predicció $\hat{y}$.

Donat un vector de característiques d'entrada X, es necessita un algorisme que pugui generar una predicció $\hat{y}$. $\hat{y}$ serà la probabilitat de què y sigui igual a 1 donades les característiques d'entrada X.

La regressió logística és un tipus d'anàlisi emprat per a predir el resultat d'una variable en funció de les variables independents o predictores, permetent modelar la probabilitat que un esdeveniment passi com a funció d'altres factors. En aquest cas, y és la variable a

predir, *X* és l'entrada i els paràmetres de regressió logística són *w* i *b*. Mentre que *b* és un nombre real, tant *X* com *w* són vectors de dimensió n_x (per a cada perceptró, cada entrada té un pes associat).

Així doncs, tornant a l'equació (6.1) es podria emprar, per exemple, la funció sigmoide com a funció d'activació per tal d'obtenir una predicció compresa entre 0 i 1. Aleshores, es defineix que un perceptró realitza dos càlculs diferenciats: primerament una regressió lineal a la qual, posteriorment, se li aplica la funció sigmoide per tal d'obtenir la predicció (Fig. 6.3 i 6.4).



$$z = w^T x + b$$

$$a = \sigma(z)$$

Fig. 6.3. Càlculs realitzats en un perceptró per obtenir la predicció $\hat{y}$.
Font: www.deeplearning.ai

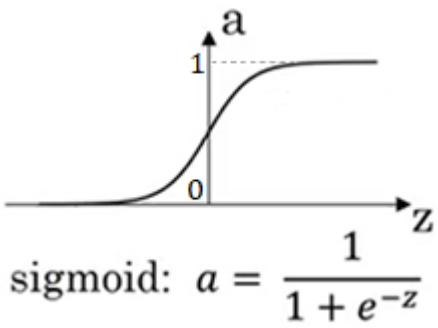


Fig. 6.4. Representació i equació de la funció sigmoide.
Font: www.deeplearning.ai

6.1.4. Obtenció dels paràmetres *w* i *b*.

Per entrenar els paràmetres *w* i *b* del model de regressió logística per tal que $\hat{y}$ esdevingui una bona estimació de la probabilitat que y sigui igual a 1, és necessari definir una funció de cost.

Tenint un conjunt d'entrenament de *m* exemples, es vol una funció que minimitzi l'error entre les prediccions $\hat{y}$ i els valors reals de les etiquetes y del conjunt d'entrenament. Aquesta funció s'anomena funció de pèrdua ($\mathcal{L}$) i es defineix per mesurar que tan bones són les sortides generades. Es podria emprar l'error quadrat, però suposaria un problema a l'hora d'optimitzar pel fet que la funció no és convexa. Per aquest motiu s'utilitza la següent funció de pèrdua de regressió logística:

$$\mathcal{L}(\hat{y}, y) = -(y \cdot \log(\hat{y}) + (1 - y) \cdot \log(1 - \hat{y})) \tag{6.3}$$

La funció de pèrdua, però, s'ha definit respecte a un únic exemple d'entrenament. És per això que es defineix la funció de cost (*J*), la qual mesura com de bé s'està realitzant un conjunt d'entrenament complet. Aquesta funció s'aplica als paràmetres *w* i *b* i serà la mitja de la suma de la funció de pèrdua aplicada a cada un dels exemples d'entrenament (i).

$$J(w, b) = \frac{1}{m} \cdot \sum_{i=1}^{m} \mathcal{L}\left(\hat{y}^{(i)}, y^{(i)}\right) \tag{6.4}$$

Així doncs, durant l'entrenament del model de regressió logística s'intenta trobar els paràmetres de pesos i *bias* que minimitzen la funció de cost *J*. Aquest problema d'optimització se soluciona mitjançant el pendent de gradient.

En la Fig. 6.5, l'eix horitzontal representa els paràmetres espacials, *w* i *b*. A la pràctica, *w* pot ser molt més gran dimensionalment, però per als fins il·lustratius es representa com un sol nombre real igual que *b*. La funció de cost *J*(*w*, *b*) és una superfície sobre el pla horitzontal *wb*. Llavors, l'alçada de la superfície representa el valor de *J*(*w*, *b*) en un cert punt i, com s'ha mencionat anteriorment, aquesta funció de cost és una funció convexa.

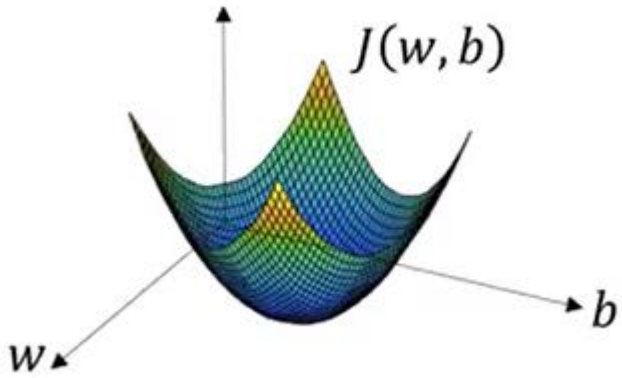


Fig. 6.5. Funció de cost *J*(*w*, *b*).
Font: www.deeplearning.ai

Per aconseguir un bon valor per als paràmetres, primerament s'inicialitzen *w* i *b* a algun valor inicial, generalment de manera aleatòria. Aleshores, el que fa el gradient és un pas des d'aquest punt inicial en la direcció de major pendent descendent. Aquest procés es repeteix fins que s'arriba a un punt proper al mínim o òptim global. L'algorisme seria el següent:

$$w = w - \alpha \cdot \frac{dJ(w,b)}{dw} \tag{6.5}$$

$$b = b - \alpha \cdot \frac{dJ(w,b)}{db} \tag{6.6}$$

Aquestes actualitzacions dels paràmetres *w* i *b* es realitzen repetidament fins que l'algorisme convergeix. α és la taxa d'aprenentatge i controla que tan gran és el pas que es fa a cada iteració del descens del gradient.

6.1.5. Aplicació a una xarxa amb diverses capes.

Fins a aquest punt, s'ha analitzat el funcionament d'un únic perceptró i com el seu model es correspon al graf de computació de la Fig.6.6, on les entrades, les característiques *X* i els paràmetres *w* i *b*, permeten calcular *z*, que després es fa servir per calcular *a* i finalment es calcula la funció de pèrdua $\mathcal{L}$. D'altra banda, s'ha fet servir *a* indistintament amb la sortida $\hat{y}$, ja que hi havia una única sortida.

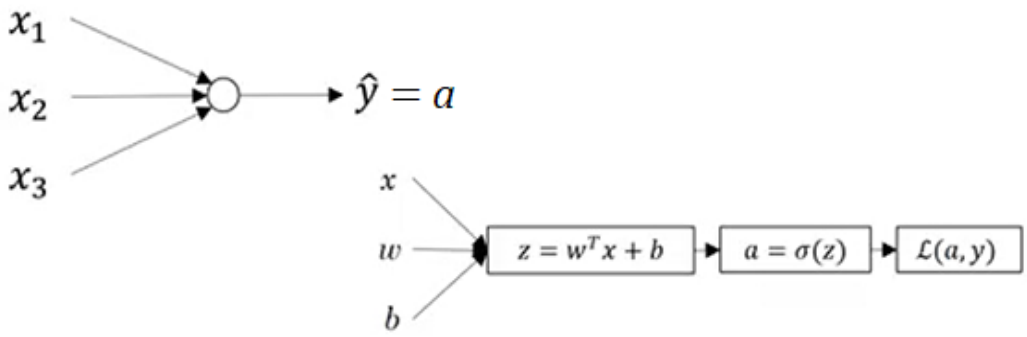


Fig. 6.6. Model i graf de computació d'un perceptró.

Font: www.deeplearning.ai

En una xarxa neuronal amb diverses capes i nodes el funcionament és similar. Per identificar cada capa s'utilitzarà el superíndex [i] (important no confondre'l amb la notació (i) emprada per a designar els exemples d'entrenament). La capa d'entrada és la 0 i no es comptabilitza, per això la xarxa de la Fig. 6.7 és una xarxa de dues capes. La capa 1 rep les entrades X i els paràmetres $w^{[1]}$ i $b^{[1]}$. Aleshores es calculen $z^{[1]}$ i $a^{[1]}$ i aquest últim passa a ser el vector d'entrada de la capa 2 juntament amb els paràmetres de pesos i *bias* associats a aquesta capa. En aquesta segona i última capa novament es calculen $z^{[2]}$ i $a^{[2]}$ i, en aquest cas, $a^{[2]}$ és la sortida final de la xarxa neuronal equivalent a $\hat{y}$. Per últim, tenint la predicció ja es pot calcular la pèrdua.

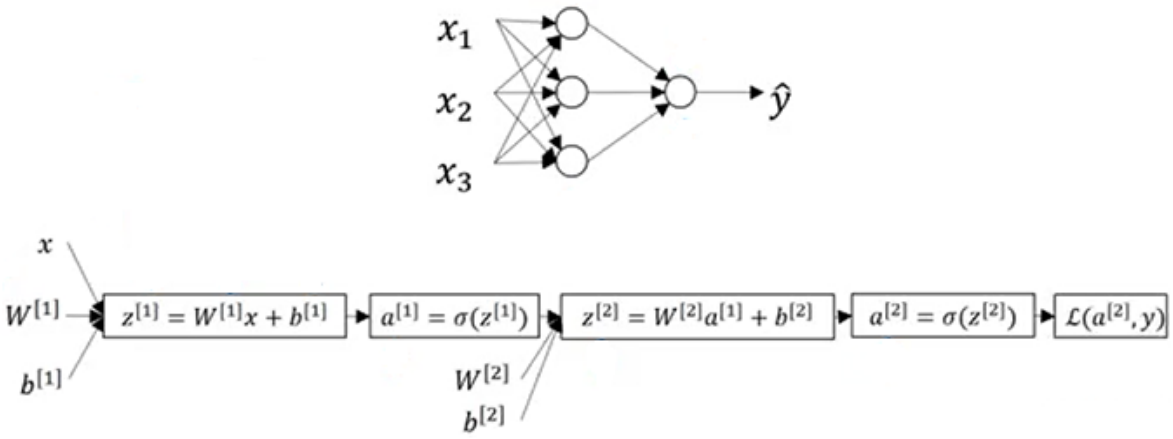


Fig. 6.7. Model i graf de computació d'una xarxa neuronal de dues capes.

Font: www.deeplearning.ai

6.1.6. Funcions d'activació.

Quan es construeix una xarxa neuronal, una de les decisions que s'ha de prendre és quina funció d'activació s'utilitzarà en les capes ocultes i quines seran les unitats de sortida de la xarxa. En els exemples anteriors s'ha utilitzat la funció d'activació sigmoide, però hi ha ocasions en què altres opcions poden funcionar millor.

A la pràctica, la funció sigmoide s'utilitza poques vegades i, normalment, sempre en la capa de sortida. Concretament s'utilitza quan es fa una classificació binària o un etiquetatge en el qual es requereix una probabilitat, casos en què es vol un valor comprès entre 0 i 1.

La funció d'activació que quasi sempre funciona millor que la funció sigmoide és la funció *tanh* o funció tangent hiperbòlica (Fig. 6.8). Matemàticament, és una versió desplaçada de la funció sigmoide perquè creui els eixos en el punt (0, 0) i escalada perquè vagi de -1 a 1. Per a les capes ocultes, si la funció *g(z)* és la *tanh(z)*, generalment funciona millor que la funció sigmoide perquè amb els valors entre 1 i -1, la mitja de les activacions que surten de la capa oculta està més a prop de ser 0. Aplicant algoritmes d'optimització, aquest fet facilita l'aprenentatge de la següent capa.

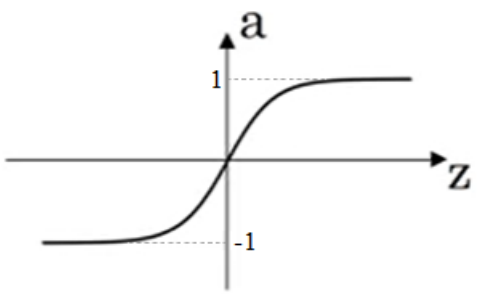


Fig. 6.8. Representació de la funció d'activació tangent hiperbòlica.
Font: www.deeplearning.ai

Un dels desavantatges tant de la funció sigmoide com de la funció *tanh* és que si *z* és molt gran o molt petita, el pendent de la funció es torna pròxim a 0. En aquests casos s'alenteix el descens de gradient, és a dir, l'aprenentatge. Una opció molt popular en l'aprenentatge automàtic és la funció *ReLU* (de l'anglès *Rectified Linear Unit*) o unitat lineal rectificada (Fig. 6.9). Sempre que *z* sigui positiva la derivada és 1 i quan *z* és negativa el pendent és 0.

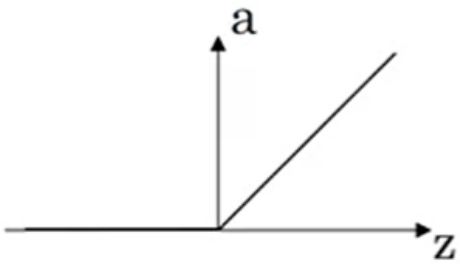


Fig. 6.9. Representació de la funció d'activació unitat lineal rectificada.
Font: www.deeplearning.ai

El fet que el pendent prengui el valor 0 novament suposa un desavantatge. Per aquest motiu hi ha una versió de la *ReLU* anomenada *ReLU* amb fuites, la qual en comptes de valer 0 quan *z* és negativa pren un lleuger pendent tal com es mostra en la Fig. 6.10.

L'avantatge tant de la funció *ReLU* com de la *ReLU* amb fuites és que, per a un gran rang de valors de *z*, la derivada de la funció d'activació és molt diferent de 0. En la pràctica, usant la funció d'activació *ReLU* una xarxa neuronal sovint aprendrà molt més ràpid que quan es fa servir el *tanh* o la funció d'activació sigmoide. Tot i que per a la meitat de la franja de *z* el pendent de *ReLU* és 0, en la pràctica una quantitat suficient de les unitats ocultes tindrà *z* més gran que 0, de tal manera que l'aprenentatge pot ser bastant ràpid per a la majoria dels exemples d'entrenament.

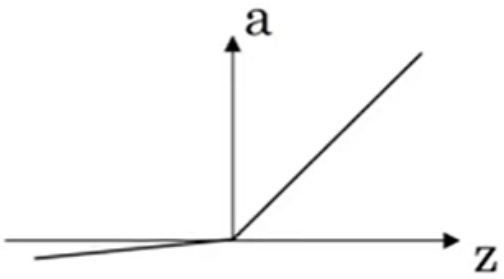


Fig. 6.10. Representació de la funció d'activació unitat lineal rectificada amb fuites.
Font: www.deeplearning.ai

6.2. Xarxes neuronals convolucionals.

Un dels desafiaments que tenen els problemes de visió per ordinador és que les entrades poden ser realment grans. Per exemple, si es treballa amb imatges de 1000 píxels per 1000 píxels, la dimensió del vector de característiques d'entrada serà de tres milions.

Però si en la primera capa oculta es tenen només 1000 unitats ocultes, el nombre total de pesos per una xarxa totalment connectada serà una matriu dimensional de 1000 per 3 milions.

Els requisits computacionals i de memòria per entrenar una xarxa neuronal amb tres mil milions de paràmetres són inviables. Per a aplicacions de visió per ordinador que utilitzen imatges grans, es necessita implementar l'operació de convolució, que és un dels pilars fonamentals de les xarxes neuronals convolucionals o CNN (de l'anglès *Convolutional Neural Network*).

6.2.1. Operació de convolució.

L'operador de convolució té l'efecte de filtrar la imatge d'entrada. Això transforma les dades de tal manera que certes característiques (determinades per la forma del filtre) es tornen més dominants en la imatge de sortida en tenir un valor numèric més alt assignat als píxels que les representen. Aquests filtres tenen habilitats de processament d'imatges específiques, com ara la detecció de vores. A continuació s'il·lustra amb un exemple.

Es té una imatge simple de 6 x 6, on la meitat esquerra de la imatge és 10 i la meitat dreta és 0. Si s'interpreta com una imatge, es podria veure que la meitat esquerra dóna els valors d'intensitat de píxels més brillants, mentre que la meitat dreta dóna valors d'intensitat de píxels més foscos. Es pot dir que en aquesta imatge clarament hi ha una vora vertical al mig (Fig. 6.11).

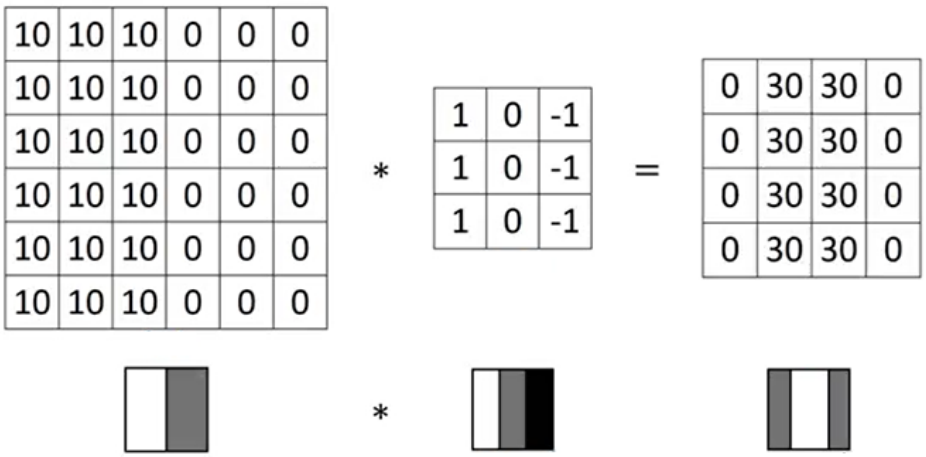


Fig. 6.11. Exemple d'aplicació de l'operació de convolució.

Font: www.deeplearning.ai

Per tal de detectar les vores verticals, es construeix un filtre representat per una matriu de 3 x 3 i s'aplica convolució amb la imatge d'entrada. El resultat d'aquest operador de convolució serà una matriu de 4 x 4.

La manera de calcular aquesta sortida és la següent. Per calcular el primer element, el superior esquerre, s'agafa el filtre i se superposa a la part superior esquerra de la imatge d'entrada original. Aleshores es multipliquen els elements d'un mateix quadrat i se sumen els resultats: 10 x 1 + 10 x 1 + 10 x 1 + 10 x 0 + 10 x 0 + 10 x 0 + 10 x -1 + 10 x -1 + 10 x -1 = 0. A continuació, per obtenir el segon element, es mou el filtre un pas a la dreta i es repeteix el procediment anterior, de tal manera que aquest cop s'obté un 30. I així successivament fins a obtenir els 16 valors. Si s'interpreta aquesta matriu resultant com una imatge, es veu una regió clara al mig. Això correspon a haver detectat aquesta vora vertical al centre de la imatge d'entrada.

Aquest exemple mostra una combinació de valors per formar un filtre detector de vores verticals, però no és l'única. La millor manera de determinar aquests nou números és tractar-los com a paràmetres que el model aprendrà en funció de la característica que vulgui detectar.

6.2.2. Padding.

Tal com s'ha vist en l'exemple anterior, si es convoluciona una imatge de 6 x 6 amb un filtre de 3 x 3, la sortida obtinguda és de 4 x 4. Aleshores si una imatge n x n es convoluciona amb un filtre f x f, llavors la dimensió de la sortida serà:

$$[n - f + 1] \text{ x } [n - f + 1] \tag{6.7}$$

Això presenta dues desavantatges. Una és que, si cada vegada que s'aplica un operador convolucional la imatge s'encongeix, només es pot aplicar unes quantes vegades abans que la imatge comenci a ser molt petita. La segona desavantatge és que els píxels en les cantonades o en les vores s'utilitzen molt menys en la sortida que els del centre, de tal manera que es perd una part de la informació de les vores de la imatge.

Per evitar-ho s'aplica el padding, que consisteix en afegir una vora addicional a la imatge generalment de zeros. Amb una vora d'un píxel, una imatge 6 x 6 passa a ser de 8 x 8 i al convolucionar-la amb un filtre 3 x 3 s'obté una sortida 6 x 6, preservant així les

dimensions originals. Si p és la quantitat de vores d'un píxel que s'afegeixen, aleshores la sortida es converteix en:

$$[n + 2p - f + 1] \text{ x } [n + 2p - f + 1] \tag{6.8}$$

En termes de quant omplir, hi ha dues opcions comuns: convolucions vàlides i convolucions iguals.

En una convolució vàlida no s'aplica *padding*. En una convolució igual, quan afegeixes vores, la mida de sortida és la mateixa que la mida d'entrada. Per saber el valor de p s'aplica la fórmula (6.9) i, per convenció, f acostuma a ser imparell.

$$p = \frac{f - 1}{2} \tag{6.9}$$

6.2.3. Pooling.

Les CNN sovint també utilitzen capes d'agrupament (en anglès *pooling*) per reduir la mida de la representació amb l'objectiu d'accelerar el càlcul, així com fer que algunes de les característiques que detecta siguin una mica més robustes.

Per exemple, es té una entrada de 4 x 4 i es vol aplicar un tipus d'agrupament anomenat agrupament màxim (en anglès *maxpooling*). Primerament es defineix la dimensió de l'agrupació, per exemple 2 x 2, i aleshores es superposa aquesta matriu a l'entrada igual que es feia amb els filtres. En aquest cas, però, el valor de sortida serà el valor màxim de la regió corresponent. A més a més, en comptes de desplaçar-lo de un en un, es pot decidir fer-ho de dos en dos, de tal manera que la imatge d'entrada queda dividida en quatre regions (Fig. 6.12). La mida d'aquest salt és un hiperparàmetre i es designa amb la lletra s (de l'anglès *stride*).

Una propietat interessant de l'agrupament màxim és que té un conjunt d'hiperparàmetres, però no té paràmetres que aprendre. Això vol dir que, una vegada fixats f i s, es realitza un càlcul fix i el descens de gradient no canvia res.

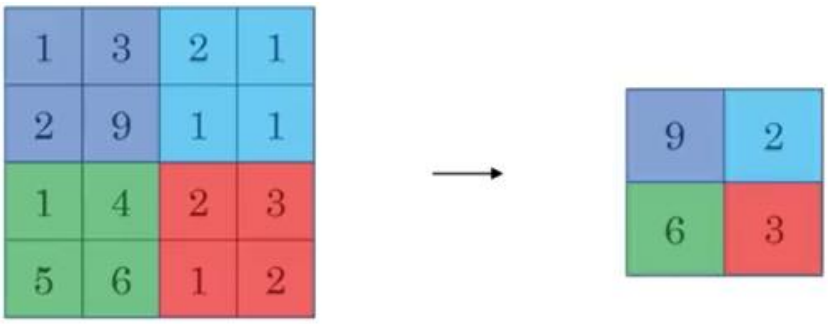


Fig. 6.12. Exemple d'aplicació d'agrupament màxim.

Font: www.deeplearning.ai

6.2.4. Convolució aplicada a imatges RGB.

Els exemples anteriors s'han realitzat amb imatges d'un sol canal, és a dir, en escala de grisos. Però tots aquests conceptes també es poden aplicar a una imatge en color. Una imatge RGB podria ser de 6 x 6 x 3, on el 3 correspon als tres canals de color. Per tal de detectar característiques en aquesta imatge, es pot convolucionar amb un filtre de 3 x 3 x 3, de tal manera que el filtre també tindrà tres capes corresponents als canals de color. La sortida, en canvi, serà una imatge de 4 x 4.

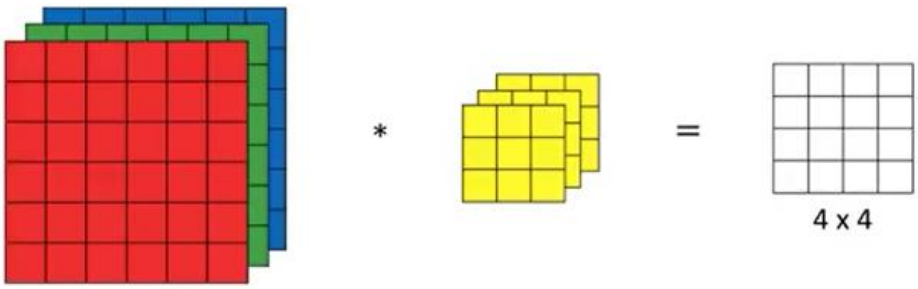


Fig. 6.13. Exemple de convolució d'una imatge RGB amb un filtre.

Fong: www.deeplearning.ai

El resultat d'aquesta operació convolucional es calcula exactament igual que es feia amb una imatge d'un canal, però amb 27 paràmetres en comptes de 9. Primerament s'agafa el filtre i es col·loca en la posició superior esquerra. Aleshores es pren cada un d'aquests 27 números i es multiplica amb els nombres corresponents als canals vermell, verd i blau de la imatge. Per últim se sumen totes aquestes xifres i s'obté el primer valor a la sortida. Aleshores, per calcular la següent sortida es desplaça el filtre cap a la dreta.

També es poden utilitzar diversos filtres al mateix temps. Per exemple, si es convoluciona la imatge d'entrada amb dos filtres 3 x 3 x 3 diferents, cada un generarà una sortida 4 x 4, i si aquestes s'apilen i es posen una rere l'altre, la sortida passa a ser una imatge 4 x 4 x 2, on 2 es correspon amb el nombre de filtres emprats.

Així doncs, un dels avantatges de l'operació de convolucó és que permet operar directament amb imatges RGB amb tres canals sense necessitat d'aplanar-les en un vector. Un altre tret a destacar és el fet que permet detectar diverses característiques diferents i la sortida tindrà un nombre de canals iguals al nombre de filtres que s'està detectant.

L'últim que cal per tenir una capa d'una xarxa neuronal convolucional és afegir un biaix i una funció d'activació no lineal a cada una de les matrius de sortida de cada un dels filtres. Una vegada aplicats aquests dos conceptes, s'apilen les diferents matrius per obtenir el volum de sortida abans mencionat.

En CNNs, és habitual trobar arquitectures que redueixen gradualment les dimensions de la imatge a mesura que n'augmenten la profunditat, és a dir, a cada capa s'incrementa el nombre de filtres a aplicar. Un cop finalitzada la fase d'aprenentatge de característiques, el volum s'aplana en un vector i es passa per un seguit de capes densament connectades per tal de realitzar la classificació. En problemes de classificació de diverses classes, la capa de sortida acostuma a utilitzar la funció d'activació *softmax*, que a diferència de la funció sigmoide que pren una entrada i li assigna una probabilitat, la funció *softmax* pot agafar múltiples entrades i assignar probabilitats per a cadascuna.

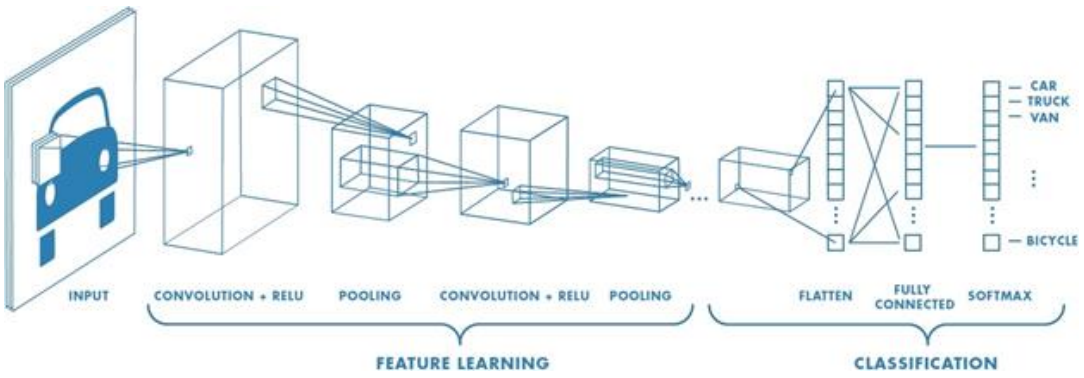


Fig. 6.14. Exemple d'arquitectura d'una xarxa neuronal convolucional.

Font: *Introducing Deep Learning with MATLAB*

6.3. Eines i tecnologia emprada.

6.3.1. Llenguatge de programació.

Entre els llenguatges de programació més utilitzats en el camp de la intel·ligència artificial destaquen especialment dos: Python i R. Però quan es tracta de *deep learning* el guanyador és Python, ja que s'han desenvolupat una gran quantitat de biblioteques i *frameworks* per a aquest llenguatge com per exemple TensorFlow o Keras.

Aquest projecte s'ha desenvolupat amb la versió 3.7.6 de Python. Tot i que ja hi ha una versió superior, aquesta encara no és compatible amb TensorFlow. Algunes de les llibreries i paquets emprats són els següents:

- Anaconda Protobuf: utilitzada per a serialitzar dades estructurades.
- Pillow: permet el processament i l'edició d'imatges des de l'entorn Python.
- lxml: emprat per a processar els fitxers XML generats durant l'etiquetatge d'imatges.
- Matplotlib: permet generar gràfiques a partir de dades expressades, per exemple, en llistes o vectors.
- Pandas: usada per a l'anàlisi de dades, principalment la generació i lectura dels arxius *csv* amb els que es treballa.
- Opencv-python: llibreria utilitzada en el processament d'imatges en temps real que ens permet treballar amb imatges, vídeos i webcams.

6.3.2. TensorFlow i Keras.

TensorFlow és una llibreria de xarxes neuronals desenvolupada per Google que ajuda a programar les arquitectures de xarxa mitjançant l'ús de diagrames de flux de dades. És de codi obert i s'ha convertit en un dels referents en el desenvolupament de sistemes d'aprenentatge profund i xarxes neuronals. S'ha utilitzat la versió 2.1.

Keras també és una llibreria de xarxes neuronals de codi obert. Està escrita en Python i és capaç d'executar-se sobre TensorFlow. De fet, l'any 2017 Keras es va integrar en el codi font de TensorFlow. Aquesta llibreria es va crear amb l'objectiu de ser amigable per a l'usuari, modular i extensible. S'ha emprat la versió 2.3.1.

6.3.3. CUDA i cuDNN.

El *deep learning* ha evolucionat considerablement amb l'ús de GPUs (*Graphics Processing Unit*). L'alt nombre de càlculs que s'efectuen i la seva elevada complexitat, especialment durant l'entrenament d'una xarxa neuronal, fan que sigui necessari un maquinari d'altes prestacions.

Les GPUs utilitzades en l'entrenament de les xarxes neuronals permeten reduir considerablement els temps de computació, en comparació a l'ús exclusiu de CPUs, gràcies a l'elevat nombre de nuclis que permeten el processament en paral·lel de multitud d'operacions.

CUDA, l'arquitectura de càlcul paral·lel de NVIDIA, juntament amb cuDNN, la seva llibreria per a xarxes neuronals profundes, permeten la utilització de GPUs en TensorFlow. En aquest projecte s'han utilitzat les versions 10.1 de CUDA i 7.6.5 de cuDNN.

6.3.4. Anaconda.

Un entorn virtual és un espai independent a la instal·lació local de l'ordinador amb l'objectiu d'aïllar els recursos i llibreries. Gràcies a aquest concepte es poden tenir diferents entorns virtuals amb diferents versions de Python o d'una llibreria concreta.

Actualment hi ha diverses alternatives per treballar amb entorns virtuals però una de les més emprades és Anaconda. Es tracta d'una distribució de codi obert, utilitzada en ciència de dades i aprenentatge automàtic, que està orientada a simplificar la gestió d'entorns i la instal·lació i administració de paquets.

6.3.5. Hardware.

El rendiment de l'aplicació pot variar en funció de les especificacions del hardware emprat, afectant als resultats finals i als temps d'execució. En aquest projecte primerament s'entrenarà el model en un ordinador i després es passarà a una Raspberry Pi 3 Model B.

L'ordinador d'entrenament està format per un processador Intel Core i5-7300 amb una freqüència base de 2,5 GHz i 8 GB de memòria RAM. D'altra banda conté una targeta gràfica NVIDIA GeForce GTX 1050 amb un total de 640 nuclis CUDA.

6.4. Preparació de l'entorn.

Es treballarà amb l'entorn virtual Anaconda. Per poder emprar la versió de TensorFlow que utilitza tant GPU com CPU s'han d'instal·lar CUDA i cuDNN.

A continuació, es crea un entorn virtual a Anaconda, s'activa i s'instal·la TensorFlow-GPU seguit dels paquets necessaris com keras, pillow, lxml, matplotlib, etc. És molt important mirar les compatibilitats entre versions de cadascun dels softwares mencionats per assegurar el correcte funcionament de l'entorn virtual generat. Una manera senzilla de comprovar que funciona correctament és obrint el prompt d'Anaconda i escrivint les següents línies:

```
python
import keras
```

Si s'han instal·lat els paquets correctament apareixerà el missatge:

```
Using TensorFlow backend.
```

6.4.1. Obtenció del *dataset*.

Després de buscar en diverses pàgines i intentar contactar sense èxit amb algunes bases de dades d'ocells com Avibase o eBird, s'ha decidit crear el *dataset* des de zero. Flickr és un portal que permet emmagatzemar, ordenar, buscar, vendre i compartir fotografies o vídeos en línia a través d'Internet.

S'ha accedit a Flickr per tal de cercar imatges dels ocells que es volen reconèixer: cadernera, pinsà borroner, passerell comú i pardal comú. Se n'ha trobat suficients per crear un *dataset*, però des del mateix portal s'han de descarregar una per una. Per tal d'evitar-ho, s'ha utilitzat l'eina Bulkr. Bulkr és una aplicació multiplataforma que està desenvolupada en Adobe Air i que permet descarregar en grans lots les imatges i àlbums de Flickr, siguin o no teves, sempre que la configuració de privacitat de les fotos que vulguis descarregar ho permetin. No permet descarregar fotos sota copyright, i clarament identifica aquelles que sí que es poden descarregar col·locant-los un logo de CC.

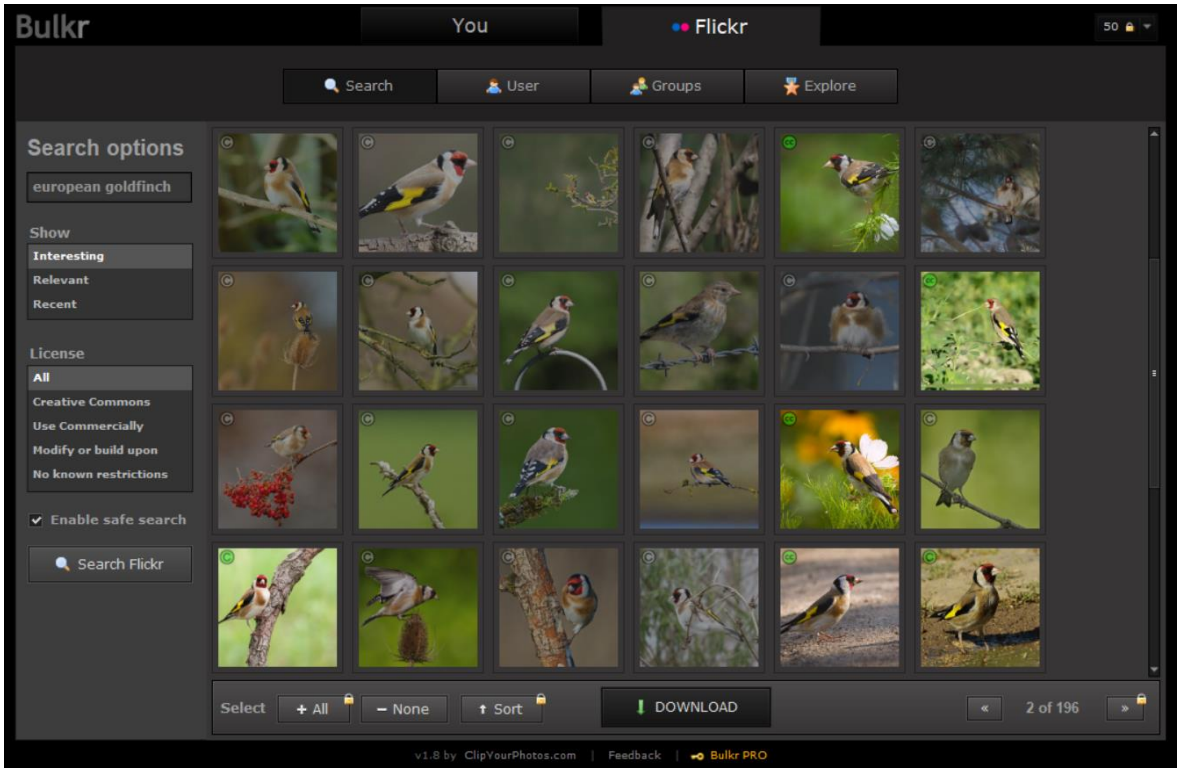


Fig. 6.15. Interfície d'usuari de l'aplicació Bulkr per a una cerca.

Mitjançant aquesta eina s'ha creat un *dataset* de 2.335 imatges en format .jpg i de dimensions iguals o inferiors als 400 x 300 píxels:

- Cadernera: 678 imatges.
- Pardal comú: 575 imatges.
- Passerell comú: 434 imatges.
- Pinsà borroner: 648 imatges.

Tot i que el pardal comú no està en perill, s'ha triat perquè és fàcil d'observar i, en el millor dels casos, es podrà fer una prova real del sistema entrenat. Com es pot observar, hi ha dos conjunts que tenen una mida més reduïda que els altres. Es vol augmentar el nombre d'imatges d'aquests perquè totes les categories estiguin més equilibrades.

Una tècnica per augmentar el volum del *dataset* és l'anomenada *data augmentation*, la qual consisteix a generar més dades d'entrenament a partir de les dades disponibles. Aplicant una sèrie de modificacions sobre una imatge original, com per exemple rotació, translació o zoom, s'obtenen diverses imatges derivades.

L'eficàcia d'aquesta tècnica resideix en la forma en què les xarxes neuronals entenen les imatges i les seves característiques, ja que si una imatge és modificada lleugerament, la xarxa la percep com una imatge completament diferent. Això disminueix les probabilitats que la xarxa se centri en orientacions o posicions mentre manté la rellevància de les característiques desitjades com poden ser les formes o els colors.

Existeixen dues maneres d'implementar *data augmentation*:

- *Offline augmentation*: en aquest cas s'augmenta la mida del conjunt de dades per un factor igual al nombre de transformacions realitzades abans d'iniciar l'entrenament de la xarxa.
- *Augmentation on the fly*: es fan les transformacions als lots amb què es va alimentant al model durant l'entrenament.

Per tal d'equilibrar el volum de les diferents classes d'imatges, s'emprarà l'augment fora de línia i es guardaran les imatges generades. S'aplicarà als *dataset* corresponents (pardal comú i passerell comú) per tal que la dimensió dels 4 conjunts sigui més similar.

Per fer-ho s'ha emprat el `ImageDataGenerator()` de Keras:

```
datagen = ImageDataGenerator(
      zoom_range = 0.2,
      rotation_range = 40,
      fill_mode = 'nearest',
      horizontal_flip = True)
```

Per a importar les imatges en lots, primerament s'ha intentat utilitzar la funció `flow_from_directory()`, però aquesta està pensada per a realitzar augments en línia, de manera que no permet guardar les imatges generades. Per aquest motiu s'ha optat per la funció `flow()`, la qual permet importar una única imatge. Per a solucionar-ho, s'han reanomenat les imatges de tal manera que formen una seqüència de números (1.jpg, 2.jpg, 3.jpg...). D'aquesta manera, mitjançant un bucle es poden anar tractant una a una:

```
input_img = 1
count = 3
group = 130

```

```
11    while input_img <= group:
12        input_path = 'data augmentation/%s.jpg' %input_img
13        image = img_to_array(load_img(input_path))
14        image = image.reshape((1,) + image.shape)
15
16        i=0
17        for batch in datagen.flow(image, batch_size=group,
18         save_to_dir = 'preview', save_format = 'jpg',
19         save_prefix = 'pard{}'.format(input_img)):
20            i += 1
21            if i >= count:
22                break
23        input_img += 1
```

La variable `group` determina quantes imatges es volen processar (la mida del lot) mentre que la variable `count` serveix per delimitar quantes imatges derivades es volen generar per cada imatge del *dataset* original.

Mitjançant aquesta pràctica s'ha augmentat el *dataset* en 345 imatges, un 15% del volum original, obtenint així un total de 2.680 fotografies per entrenar la xarxa neuronal. Incloent les imatges generades, es tenen 680 imatges de pardal comú i 674 de passerell comú.

Fig. 6.16. Imatges originals (esquerra) i quatre imatges generades amb *data augmentation*.

6.5. Xarxa neuronal simple.

Primerament es crea una xarxa neuronal simple de 3 capes i s'analitzen els resultats.

6.5.1. Programació.

Abans de començar a programar, però, per tal de poder importar les imatges és important que el directori on es guarden les imatges segueixi la següent estructura:

```
C:.
├───ocells
    ├───Cadernera
    ├───Pardal_comu
    ├───Passerell_comu
    └───Pinsa_borroner
```

Dins del directori \ocells, hi ha tres subcarpetes anomenades igual que les diferents classes: \Cadernera, \Pardal_comu, \Passerell_comu i \Pinsa_borroner. En cadascun d'aquests directoris hi ha les imatges pertanyents a les respectives classes.

A continuació, es crea un fitxer anomenat *simple_nn.py* i es comença a programar.

```
 1   data = []
 2   labels = []
 3
 4   imagePaths = sorted(list(paths.list_images('ocells')))
 5
 6   for imagePath in imagePaths:
 7       image = cv2.imread(imagePath)
 8       image = cv2.resize(image, (32, 32)).flatten()
 9       data.append(image)
10       label = imagePath.split(os.path.sep)[-2]
11       labels.append(label)
```

Després d'importar les llibreries necessàries, s'inicialitzen les llistes `data` i `labels` (línies 1 i 2) i es determina el *path* fins al directori del *dataset* (línia 4). Concretament es fa una llista dels *path* de cadascuna de les imatges que hi ha dins el directori \ocells. A continuació, per a cadascun d'aquests *path*, es carrega la imatge corresponent, es redimensiona a 32 x 32 píxels i es converteix en un vector amb la funció `flatten()` (línies 7 i 8). Per últim es guarda el vector en la llista `data` (línia 9).

Un cop guardada la imatge com a vector, s'extreu el nom de la carpeta on estava la fotografia, la qual coincideix amb la classe a la qual pertany la imatge si s'ha seguit l'estructura prèviament mencionada, i es guarda en la llista `label` (línies 10 i 11).

```
13    data = np.array(data, dtype="float") / 255.0
14    labels = np.array(labels)
```

Finalment, es converteixen ambdues llistes en *arrays* i els valors de les intensitats dels píxels s'escalen del rang original [0, 255] a [0, 1]. Ja només falta dividir les dades entre *test* i *train*. El primer conjunt és el que s'utilitza per a entrenar la xarxa neuronal, mentre que el segon s'empra per tal d'avaluar el funcionament del model.

```
16    (trainX, testX, trainY, testY) = train_test_split(data,
17         labels, test_size=0.25, random_state=42)
18
19    lb = LabelBinarizer()
20    trainY = lb.fit_transform(trainY)
21    testY = lb.transform(testY)
```

Mitjançant la funció `train_test_split()` es divideix el *dataset* en dos conjunts, destinant un 75% de les imatges al *train* i el 25% restant al *test*, i s'empra el paràmetre `random_state` per assegurar que, per cada experiment que es realitzi, aquests dos conjunts seran sempre els mateixos. Tant `trainX` com `testX` formen les dades, mentre que `trainY` i `testY` contenen les etiquetes. Aquestes etiquetes estan representades com un *array* de *strings*, però per a passar-les al model de Keras han de ser *integers* representats per un vector de codificació única (un vector de zeros amb un 1 en la columna corresponent a la classe en qüestió). Per fer-ho s'empra la funció `LabelBinarizer()` en les línies 19 – 21.

Després d'importar les imatges i preparar-les per poder utilitzar-les, es procedeix a definir l'estructura del model.

```
23    model = Sequential()
24    model.add(Dense(1024, input_shape=(3072,),
25         activation="tanh"))
26    model.add(Dense(512, activation="tanh"))
27    model.add(Dense(4, activation="softmax"))
```

Primerament es defineix que el model és seqüencial (línia 23), és a dir, està format per diverses capes apilades una rere l'altra. A continuació, es defineixen la capa d'entrada i la primera capa oculta a la línia 24. La capa d'entrada té una dimensió de 3072, ja que les imatges són de 32 x 32 x 3 = 3027 píxels, i la primera capa oculta està formada per 1024 nodes. La segona capa oculta està formada per 512 perceptrons i finalment la capa de sortida té únicament 4 unitats, les quals es corresponen amb les quatre classes.

Les capes ocultes utilitzen la funció d'activació *tanh*, mentre que la de sortida empra la *softmax* per a realitzar la classificació, ja que aquesta . El fet que les capes siguin denses significa que cada neurona rep entrada de totes les neurones de la capa anterior.

Amb aquestes 5 línies de codi s'ha creat una xarxa neuronal de 3 capes. Abans d'entrenar-la, s'ha de configurar el procés d'aprenentatge, la qual cosa es fa mitjançant la consigna `compile()`.

```
29    opt = SGD(lr=0.01)
30    model.compile(loss="categorical_crossentropy",
31          optimizer=opt,metrics=["accuracy"])
```

Es defineix el descens de gradient (*Stochastic Gradient Descent*) com a optimitzador amb una taxa d'aprenentatge (*Learning Rate*) de 0,01. A continuació s'indica que la funció de pèrdua és la `categorical crossentropy`, la qual s'utilitza quan l'objecte a detectar pot pertànyer a una única classe, i que la funció a emprar per jutjar el rendiment del model és la precisió.

```
33    H = model.fit(trainX, trainY, validation_data=(testX,
34          testY),epochs=75, batch_size=32)
```

En les línies 33 i 34 s'entrena el model mitjançant la consigna `fit()`. S'utilitza el conjunt d'imatges de *train* i les corresponents etiquetes per a l'entrenament, mentre que el conjunt *test* serveix per avaluar com va evolucionant l'aprenentatge del model. El paràmetre `batch_size` controla nombre de mostres a processar per actualització del gradient i el paràmetre `epochs` indica el nombre d'iteracions a realitzar sobre el conjunt de dades d'entrenament.

Finalment, s'avalua el funcionament del model, es genera un *plot* de l'evolució de la precisió i les pèrdues durant el procés d'aprenentatge i es guarda el model ja entrenat.

6.5.2. Resultats.

Un cop executat el codi, l'entrenament dura aproximadament 30 segons. A continuació es genera un informe sobre l'avaluació realitzada amb el conjunt *test*.

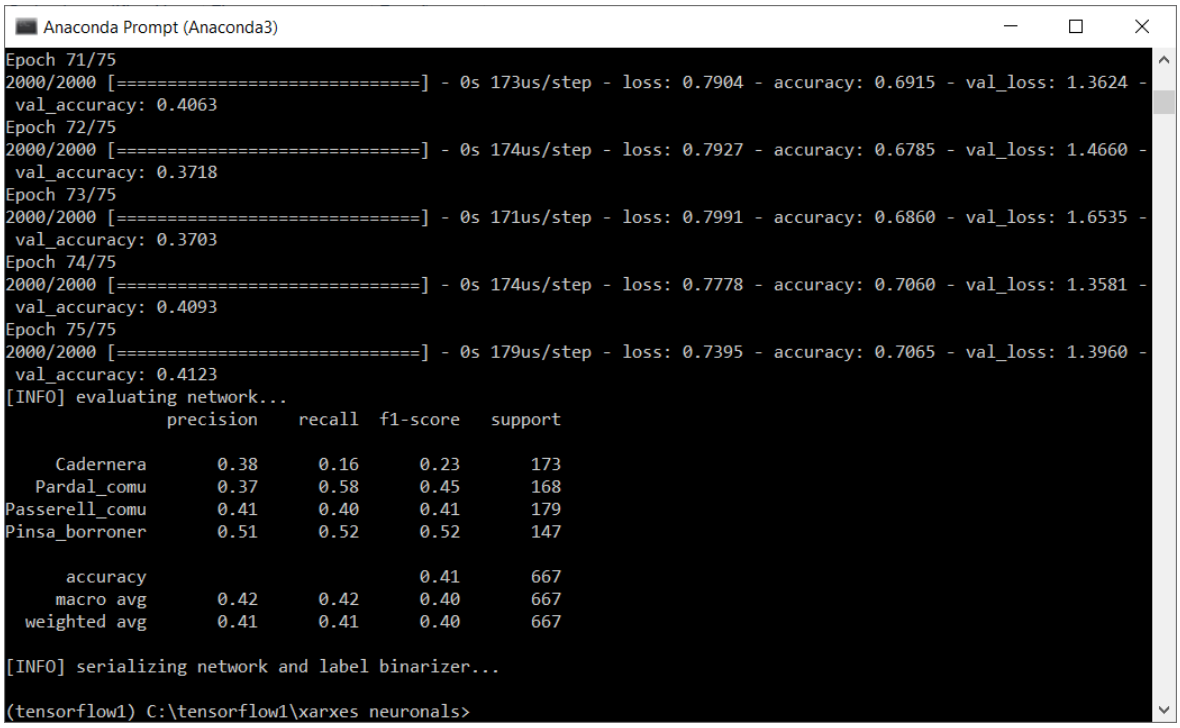


Fig. 6.17. Informe de l'avaluació de la xarxa neuronal simple.

En aquest informe (Fig. 6.17) es pot observar que la precisió del model és del 41%. Tenint en compte que les probabilitats d'encertar de manera aleatòria són del 25%, es pot afirmar que la xarxa neuronal ha après patrons que es poden utilitzar per discriminar les quatre classes.

D'altra banda, s'ha generat un gràfic amb els valors de precisió tant de l'entrenament (train_acc) com de l'avaluació (val_acc), i els valors de pèrdua d'ambdós processos (train_loss i val_loss respectivament).

En la Fig. 6.18 es pot observar que clarament s'ha produït *overfitting*. Aquest terme fa referència a quan un model aprèn característiques d'un exemple concret no rellevants en la classe, de tal manera que no és capaç de generalitzar amb futures dades. En el gràfic es pot detectar perquè les pèrdues de l'entrenament i la validació arriba un punt en què divergeixen i se separen considerablement.

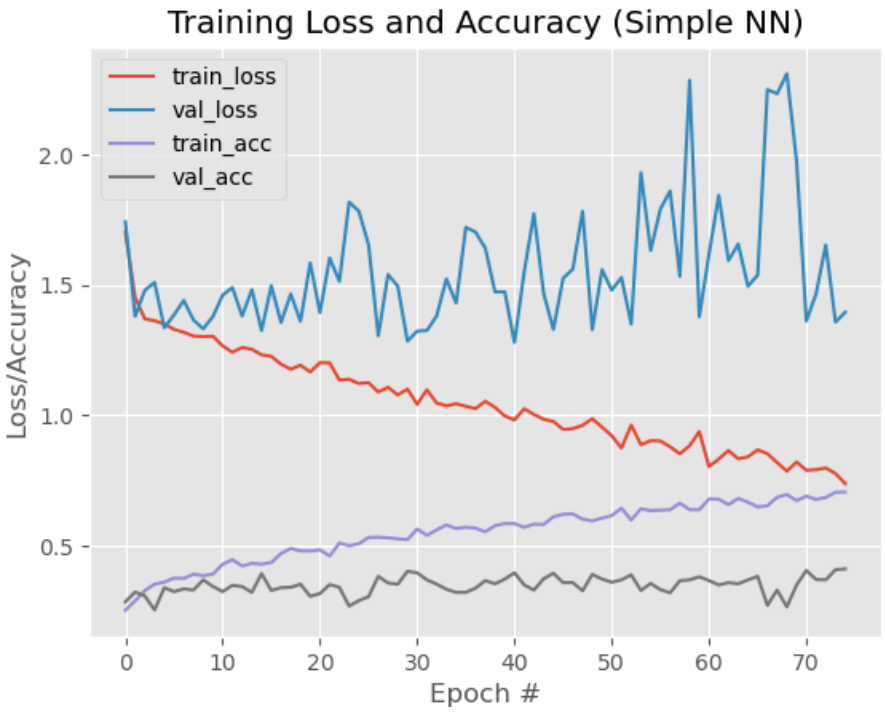


Fig. 6.18. Gràfic de la precisió i la pèrdua durant l'entrenament i la validació de la xarxa neuronal simple.

6.6. Xarxa neuronal convolucional.

A continuació es programa una xarxa neuronal convolucional. Es decideix entrenar-la de tres maneres diferents per tal d'intentar millorar progressivament els resultats. Primerament s'entrena amb el mateix *dataset* que s'ha emprat per a la xarxa simple. En el segon cas s'afegeix *data augmentation on the fly*. Per últim, es prova d'augmentar les dimensions de les imatges d'entrada.

El codi del model es desenvolupa en un fitxer diferent al dels entrenaments i tots ells l'importen.

6.6.1. Programació del model.

Per desenvolupar el model s'ha pres com a referència l'arquitectura VGG16. Aquesta arquitectura va ser introduïda per K. Simonyan i A. Zisserman de la Universitat d'Oxford en el seu treball de 2014 *Very Deep Convolutional Networks for Large-Scale Image Recognition*. Es caracteritza per la seva senzillesa, utilitzant només capes convolutives amb filtres 3 x 3 apilades les unes rere les altres i augmentat cada cop més la profunditat. La

reducció de la mida del volum es gestiona amb capes *max pooling*. Finalment, es col·loquen dues capes completament connectades, cadascuna amb 4.096 nodes, seguides d'un classificador *softmax* (Fig. 6.19).

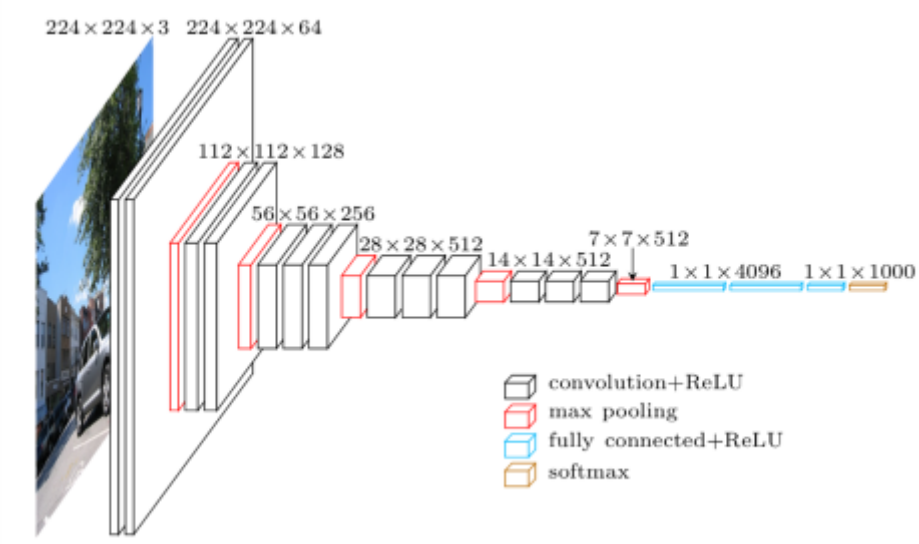


Fig. 6.19. Representació de l'arquitectura VGG16.

Font: www.mc.ai

Un dels problemes que presenta, però, és que s'entrena molt lentament. Per aquest motiu, s'ha optat per fer una versió més reduïda d'aquesta arquitectura.

Es crea un nou fitxer anomenat *smallvggnet.py* i, després d'importar les llibreries necessàries, es comença a desenvolupar el model.

```
1   class SmallVGGNet:
2       @staticmethod
3       def build(width, height, depth, classes):
4           model = Sequential()
5           inputShape = (height, width, depth)
```

Primerament es defineix la classe `SmallVGGNet`, la qual s'importarà en els fitxers d'entrenament. També es defineixen els paràmetres necessaris per a utilitzar-la: alçada i profunditat de les imatges, així com el nombre de canals i de classes (línia 3).

A continuació, s'inicialitza el model seqüencial i es crea una variable amb la mida de les imatges d'entrada (línia 5).

```
 7              model.add(Conv2D(32, (3, 3), padding="same",
 8                   input_shape=inputShape))
 9              model.add(Activation("relu"))
10              model.add(BatchNormalization(axis=-1))
11              model.add(MaxPooling2D(pool_size=(2, 2)))
12              model.add(Dropout(0.25))
```

La primera capa convolucional està formada per 32 filtres 3 x 3 (línia 7). D'altra banda s'especifica la mida de les imatges d'entrada i s'empra *same padding* per tal que les dimensions de la matriu de sortida siguin les mateixes que les d'entrada.

Aquesta arquitectura de xarxa utilitza la funció d'activació *ReLU*. S'aplica una capa *max pooling* per tal de reduir la mida progressivament i s'utilitza *batch normalization*. La normalització per lots s'utilitza per normalitzar les activacions d'un volum d'entrada determinat abans de passar-lo a la següent capa de la xarxa, és a dir, aplica una transformació que manté l'activació mitjana propera a 0 i la desviació estàndard d'activació propera a 1.

Finalment s'aplica una capa de *dropout*. Aquest procés consisteix a anar desconnectant neurones de manera aleatòria entre capes. L'objectiu és fer que la xarxa sigui més robusta, reduint l'*overfitting* i augmentant la precisió.

```
14              model.add(Conv2D(64, (3, 3), padding="same"))
15              model.add(Activation("relu"))
16              model.add(BatchNormalization(axis=-1))
17              model.add(Conv2D(64, (3, 3), padding="same"))
18              model.add(Activation("relu"))
19              model.add(BatchNormalization(axis=-1))
20              model.add(MaxPooling2D(pool_size=(2, 2)))
21              model.add(Dropout(0.25))
```

El següent bloc està format per dues capes convolucionals de 64 filtres, novament conservant les dimensions d'entrada, seguides per un *max pooling* i un *dropout*. D'aquesta manera la reducció de la mida és gradual i controlada.

```
23              model.add(Conv2D(128, (3, 3),padding="same"))
24              model.add(Activation("relu"))
25              model.add(BatchNormalization(axis=-1))
26              model.add(Conv2D(128, (3, 3),padding="same"))
```

```
27              model.add(Activation("relu"))
28              model.add(BatchNormalization(axis=-1))
29              model.add(Conv2D(128, (3, 3),padding="same"))
30              model.add(Activation("relu"))
31              model.add(BatchNormalization(axis=-1))
32              model.add(MaxPooling2D(pool_size=(2, 2)))
33              model.add(Dropout(0.25))
```

L'últim bloc de capes convolucionals està format per 3 capes de 128 filtres. La mida dels filtres se segueix mantenint i la reducció de mida segueix sent controlada.

```
35              model.add(Flatten())
36              model.add(Dense(512))
37              model.add(Activation("relu"))
38              model.add(BatchNormalization())
39              model.add(Dropout(0.5))
40              model.add(Dense(classes))
41              model.add(Activation("softmax"))
42
43              return model
```

Per acabar, el volum es converteix en un vector (línia 29) que passa a ser l'entrada d'una capa densa de 512 perceptrons (línia 30). Finalment s'afegeix la capa de sortida formada per tants nodes com classes a reconèixer (línia 33) activada mitjançant la funció *softmax*, i es retorna l'arquitectura de la xarxa.

6.6.2. Entrenament simple.

Aquest primer entrenament consisteix a emprar el matix *dataset* que ja s'ha utilitzat per entrenar la xarxa neuronal simple. Es crea un fitxer anomenat *train_vgg.py*.

El codi per tal d'importar les imatges i generar els conjunts de *test* i *train* és exactament el mateix. Únicament, en aquest cas les dimensions de les imatges s'ha fixat a 64 x 64 en comptes de 32 x 32.

```
1   from smallvggnet import SmallVGGNet
2
3   model = SmallVGGNet.build(width=64, height=64, depth=3,
4         class=4)
```

A la línia 1 s'importa la classe creada en el fitxer *smallvggnet.py* creat anteriorment. Aleshores s'inicialitza el model passant-li els paràmetres requerits (línies 3 i 4).

A continuació, es configura el procés d'aprenentatge (`compile()`) novament amb el descens de gradient com a optimitzador, una taxa d'aprenentatge de 0,01 i la funció de pèrdua `categorical crossentropy`. Finalment s'entrena el model (`fit()`), s'avalua i es guarda el model entrenat juntament amb el plot de la precisió i les pèrdues. Aquestes línies de codi també són les mateixes que les emprades amb la xarxa anterior.

6.6.3. Entrenament amb data augmentation on the fly.

Per a realitzar aquest entrenament, s'agafa el fitxer *train_vgg.py*, es realitza una còpia i es reanomena *train_vgg_da.py*.

En aquest nou fitxer, abans d'inicialitzar el model es prepara el generador d'imatges.

```
1   aug = ImageDataGenerator(rotation_range=30,
2         width_shift_range=0.1, height_shift_range=0.1,
3         shear_range=0.2, zoom_range=0.2,
4         horizontal_flip=True, fill_mode="nearest")
```

S'utilitza la funció `ImageDataGenerator()` vista prèviament en el *data augmentation offline*. En aquest cas, s'han afegit el desplaçament horitzontal i vertical.

Un cop preparat el generador, s'ha de cridar la funció per tal de realitzar l'augment de dades durant el procés d'entrenament. Així doncs, es canvia la línia de codi on es defineix l'entrenament `H`.

```
6   H = model.fit_generator(aug.flow(trainX, trainY,
7         batch_size=32), validation_data=(testX, testY),
8         steps_per_epoch=len(trainX) // 32, epochs=75)
```

Es mantenen els valors de `batch_size` i `epochs`, però s'hi afegeixen els `steps_per_epoch`, que es corresponen amb el nombre de dades del conjunt *train* entre la mida del lot, i es fa el *data augmentation* configurat en la variable `aug` amb les imatges d'entrenament.

En aquest cas, el *data augmentation* es realitza durant l'entrenament, de tal manera que les imatges es generen durant el procés i s'utilitzen sense necessitat de guardar-les. La resta de codi es manté.

6.6.4. Entrenament amb imatges 200 x 200

En aquest entrenament, es manté el codi de l'apartat anterior incrementant únicament la mida de les imatges. El fitxer s'anomena *train_vgg_da_200.py*.

Aquest paràmetre està definit durant la importació i redimensionat de les imatges.

```
1    for imagePath in imagePaths:
2        image = cv2.imread(imagePath)
3        image = cv2.resize(image, (200, 200)).flatten()
4        data.append(image)
```

També és un dels paràmetres necessaris per inicialitzar el model.

```
6    model = SmallVGGNet.build(width=200, height=200,
7          depth=3, class=4)
```

Aquest últim entrenament, doncs, manté el *data augmentation on the fly* alhora que importa les imatges amb una mida superior.

6.6.5. Resultats.

A mesura que s'intenta millorar la precisió del model, el temps d'entrenament també augmenta. Aquest fet es veu reflectit en la Taula 6.1, on DA significa *Data Augmentation*.

Entrenament	**Simple**	**DA on the fly**	**Imatges 200 x 200**
Precisió	82%	86%	90%
Temps d'entrenament	30 s	5 min	45 min

Taula 6.1. Relació entre la precisió del model i el temps d'entrenament.

En comparació amb la xarxa neuronal simple, amb la qual s'obtenia una precisió del 41%, el fet de canviar l'estructura del model a una xarxa convolucional ha duplicat la precisió.

En aplicar *data augmentation on the fly*, tot i augmentar breument el temps d'entrenament, també s'ha aconseguit millorar la precisió del model arribant al 86%.

Per acabar, el fet d'augmentar la mida de les fotografies d'entrada de 64 x 64 a 200 x 200 píxels augmenta considerablement el temps d'entrenament fins a tres quarts d'hora, quan l'entrenament simple es realitza en mig minut. Tot i això, aquest entrenament és el que ha donat millors resultats, obtenint una precisió del 90%.

La Fig. 6.20 mostra com, durant l'entrenament i validació de CNN, a mesura que disminueixen les pèrdues de l'entrenament també ho fan les de la validació, obtenint un valor final més petit que el del gràfic de la xarxa simple. Així mateix, la precisió d'ambdós processos també incrementa paral·lelament, aconseguint un millor valor final.

En aplicar *data augmentation on the fly* (Fig. 6.21), les corbes de precisió deixen de ser paral·leles per anar, pràcticament, superposades l'una a l'altra. El mateix succeeix amb les corbes de pèrdues, tot i que les de validació mostren pics pronunciats en comparació amb les d'entrenament.

Finalment, en la Fig. 6.22 s'aprecia com les corbes de precisió s'apropen més al valor 1 que en els casos anteriors. Les corbes de pèrdues, però, tornen a paral·lelitzar-se i els pics de validació augmenten, sobretot durant les primeres iteracions.

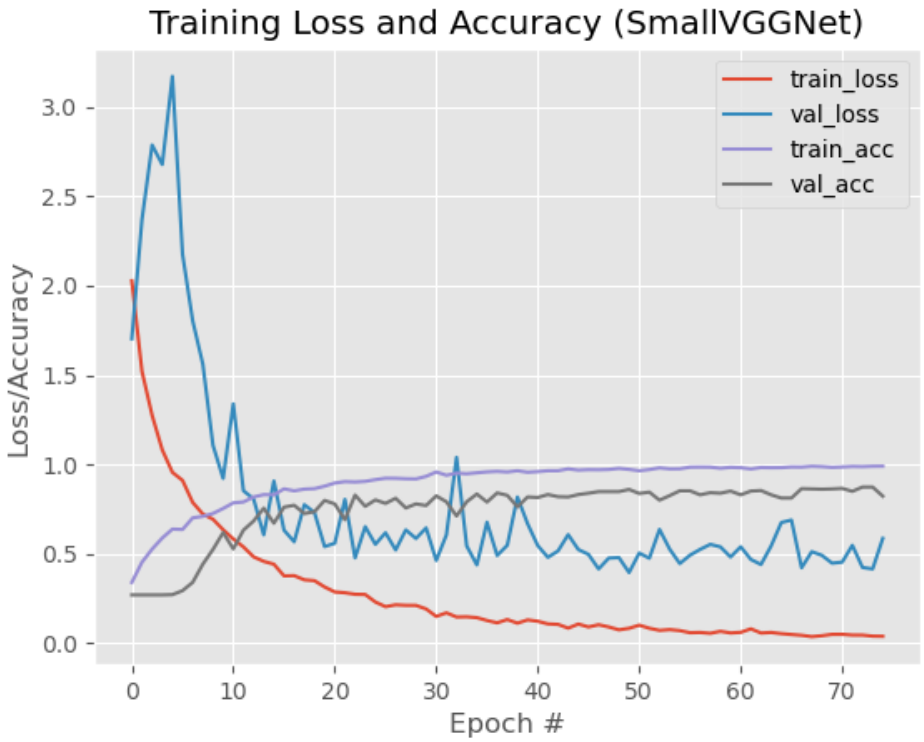


Fig. 6.20. Gràfic de la precisió i la pèrdua durant l'entrenament i la validació de la CNN.

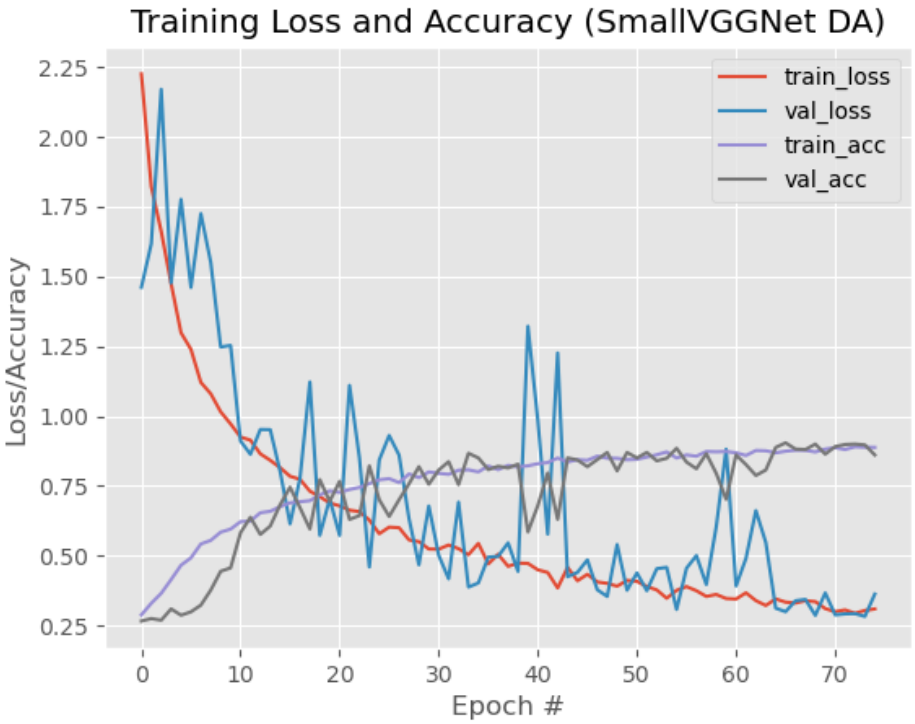


Fig. 6.21. Gràfic de la precisió i la pèrdua durant l'entrenament i la validació de la CNN aplicant *data augmentation on the fly*.

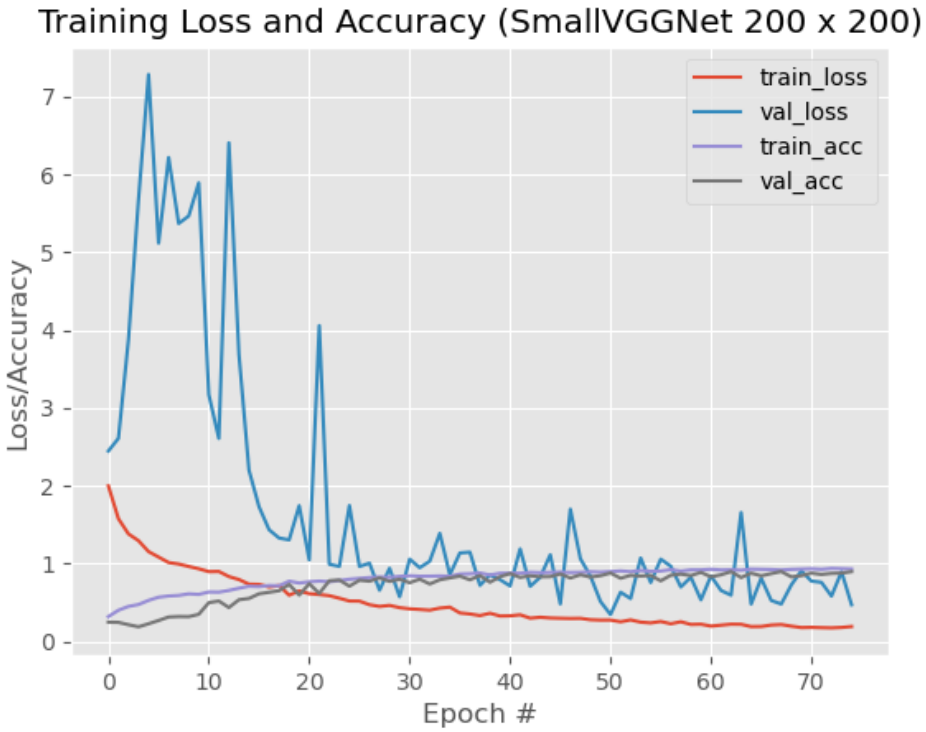


Fig. 6.22. Gràfic de la precisió i la pèrdua durant l'entrenament i la validació de la CNN augmentat la mida de les imatges d'entrada a 200 x 200 píxels.

En tots tres casos s'ha entrenat la mateixa xarxa, però s'ha variat el volum i les característiques de les dades d'entrada, veient com aquests paràmetres afecten de manera important als resultats finals. Per comprovar que l'últim model és un bon model, s'avalua (Fig. 6.23) i es genera una matriu de confusió (Fig. 6.24).

	precision	recall	f1-score	support
Cadernera	0.97	0.89	0.93	173
Pardal_comu	0.87	0.83	0.85	168
Passerell_comu	0.84	0.94	0.89	179
Pinsa_borroner	0.93	0.94	0.93	147
accuracy			0.90	667
macro avg	0.90	0.90	0.90	667
weighted avg	0.90	0.90	0.90	667

Fig. 6.23. Informe de l'avaluació de la CNN amb imatges d'entrada de 200 x 200 píxels.

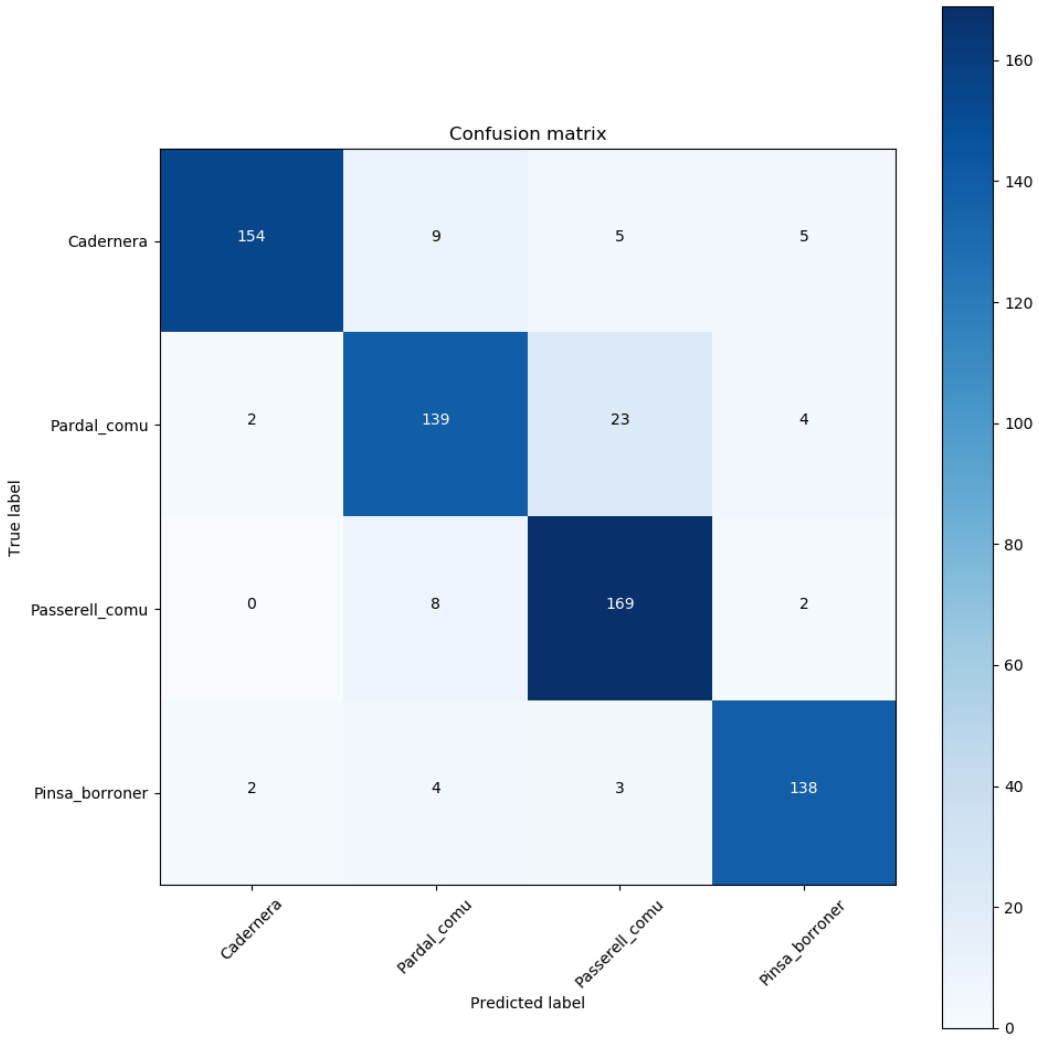


Fig. 6.24. Matriu de confusió de l'avaluació de la Fig. 6.23.

L'informe mostra que el F1-score és proper al 90% per a les quatre classes i en la matriu de confusió s'aprecia clarament una diagonal més fosca reflectint els encerts del model. Únicament prediu alguns pardals comuns com a passerells comuns, però tot i això el nombre d'errors comesos és molt reduït en comparació amb el d'encerts.

6.7. Aplicació de transfer learning.

Existeixen dues tècniques per aplicar *transfer learning*. Una d'elles consisteix a congelar l'estructura i els pesos de la base convolucional, formada per les capes pertanyents a la fase d'aprenentatge de característiques, i substituir les capes de classificació per unes de pròpies. D'aquesta manera el nombre de paràmetres que s'entrena és menor i s'utilitza el model pre-entrenat per extreure les característiques generals. Keras conté diversos models els quals es poden importar sencers o sense les *top layers*, és a dir, sense les capes de classificació.

D'altra banda, també es poden congelar només algunes capes de la base convolucional en comptes de totes elles. Aquesta tècnica es fa servir sobretot quan el conjunt de dades és petit i diferent del conjunt de dades del model prèviament entrenat. Tot i que pot ser complicat trobar un equilibri entre el nombre de capes a congelar i el nombre de capes a entrenar, l'API de TensorFlow permet aplicar aquest mètode de manera senzilla, ja que no mostra l'arquitectura del model i pren ella aquesta decisió. Per contra, utilitzar-la requereix més feina que els models de Keras perquè s'han de preparar les imatges prèviament i generar uns fitxers concrets per a poder entrenar el model.

S'ha optat per emprar l'API de TensorFlow perquè el volum de dades de què es disposa és reduït i no es volen detectar diversos objectes completament diferents entre si (persones, cotxes, pilotes...) com en el model original, sinó que les diverses classes a detectar poden resultar semblants i es busquen característiques molt específiques de cada espècie.

6.7.1. Preparació de l'entorn.

L'API de detecció d'objectes de TensorFlow requereix utilitzar una estructura de directori específica. A la web GitHub es pot trobar el directori amb l'estructura ja creada i l'API inclosa, únicament s'ha de descarregar [13]. Aleshores es crea una carpeta en el disc C: anomenada "tensorflow1" que serà el directori de treball i en la qual es guarda l'arxiu

descarregat un cop s'ha descomprimit. Aquest directori conté diversos models, en aquest projecte es treballarà amb el model de detecció d'imatges, ubicat en C:\tensorflow1\models\research\object_detection.

A continuació es descarrega el model de xarxa neuronal amb el qual es vol treballar. Per a triar-lo s'ha tingut en compte que TensorFlow Lite no admet models RCNN, únicament admet models SSD. TensorFlow Lite és una solució lleugera de TensorFlow per a dispositius mòbils i integrats que permet executar models de *machine learning* en dispositius de baixa latència, i serà la versió de TensorFlow utilitzada per a executar el model finalment a la RPi. El model triat ha sigut SSD-MoileNet-V2-Quantized-COCO. Aquest es descarrega i es desa a la carpeta \object_detection del directori de treball.

Finalment, es compilen els fitxers Protobuf que TensorFlow utilitza per configurar el model i els paràmetres d'entrenament. D'aquesta manera es genera un fitxer .py per a cada fitxer .proto.

6.7.2. Preparació de les imatges.

Una vegada obtingut el *dataset*, les imatges s'han de preparar per poder utilitzar-les per a entrenar la xarxa neuronal. Primerament es reparteixen les imatges en dos directoris: un d'entrenament i un d'avaluació. Un 80% de les imatges es mouen al directori \object_detection\images\train i el 20% restant al directori \object_detection\images\test, procurant que les imatges de cada carpeta siguin variades. A continuació, cada element d'entrenament x_i ha de tenir associat un valor de sortida y. Per això, s'han d'etiquetar les imatges. Aquest procés consisteix a, amb un programa concret com pot ser LabelImg, dibuixar un rectangle al voltant de l'objecte a reconèixer i posar-li una etiqueta o nom per a identificar-lo (Fig. 6.25). Un cop realitzades les etiquetes de la fotografia es prem el botó de guardar, de tal manera que el programa genera un fitxer .xml amb el mateix nom que la imatge el qual conté els paràmetres de l'etiqueta realitzada. Aquests passos es repeteixen per a cada fotografia tant del directori \train com del \test.

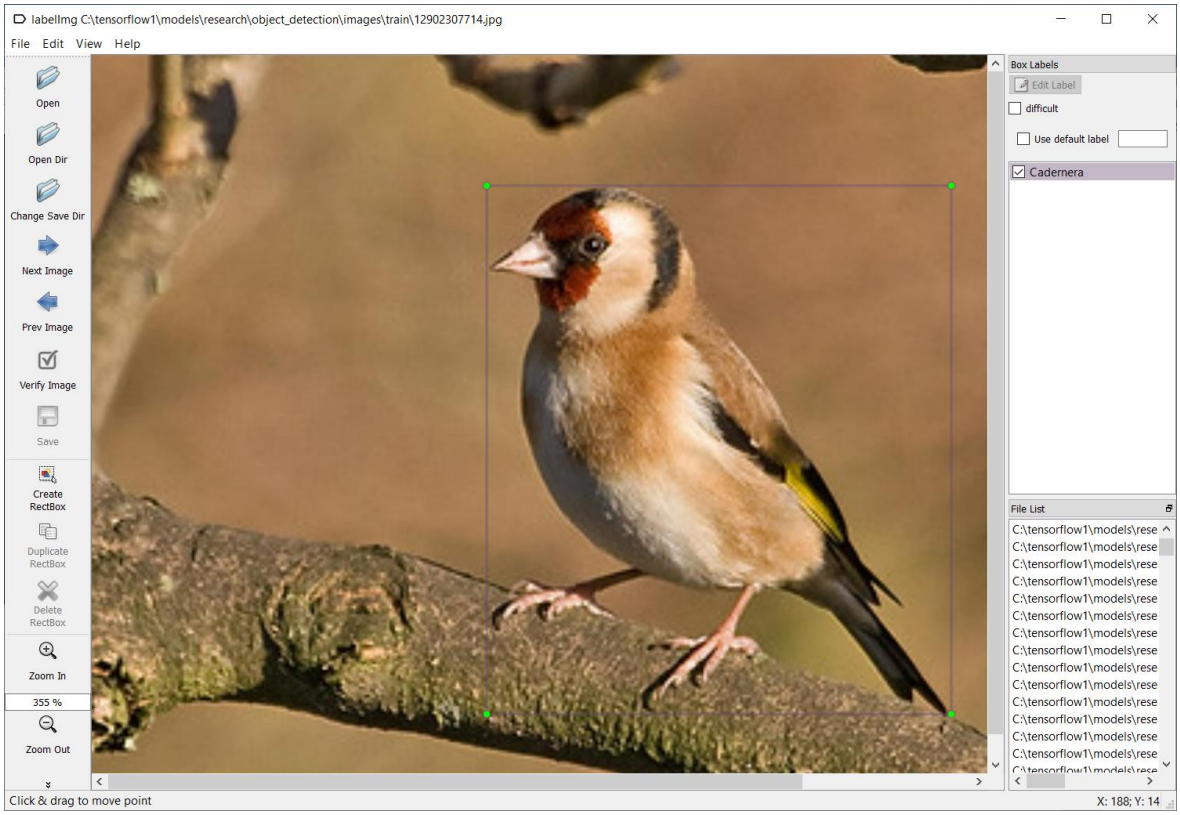


Fig. 6.25. Exemple d'etiquetatge d'una imatge mitjançant el programa LabelImg.

Quan s'ha completat el procés d'etiquetatge de les 2.680 imatges que conformen el *dataset*, es procedeix a escalar-les. Aquest pas és important per tal que totes les entrades de la xarxa neuronal tinguin la mateixa mida, és a dir, que les dimensions del vector de característiques d'entrada *X* sigui fixa. En aquest projecte, però, l'escalat de les imatges està incorporat en el codi del model a entrenar, concretament en les línies 43 - 47 de l'arxiu de configuració, de tal manera que s'omet aquest pas.

6.7.3. Generació de les dades d'entrenament.

Un cop preparades les imatges, s'han de generar els fitxers que s'empraran per entrenar el model.

Primerament, mitjançant el fitxer *xml_to_csv.py* ubicat \object_detection, es genera un fitxer *train_labels.csv* que conté la informació de tots els fitxers .xml del directori \train. El mateix es fa amb els fitxers del directori \test, creant un fitxer *test_labels.csv*.

A continuació, s'obre el document *generate_tfrecord.py* i s'edita el mapa d'etiquetes, incloent-hi les que ha de detectar el model:

```
def class_text_to_int(row_label):
    if row_label == 'Cadernera':
        return 1
    elif row_label == 'Pinsa borroner':
        return 2
    elif row_label == 'Passerell comu':
        return 3
    elif row_label == 'Pardal comu':
        return 4
    else
        None
```

Aquest fitxer s'utilitza per generar els anomenats *TFRecord*. Aquest nom fa referència a un format d'arxiu de TensorFlow i un dels avantatges que presenta és que permet llegir més ràpidament les dades durant l'entrenament. Aleshores s'executa *generate_tfrecord.py* des del prompt dues vegades, la primera passant-li el fitxer .csv amb les dades de *train* i la segona amb les de *test*. D'aquesta manera s'obtenen dos arxius nous: *train.record* i *test.record*, que són els que utilitzarà el model.

Ara que ja estan les dades llestes perquè el model les pugui utilitzar, únicament queda crear un mapa d'etiquetes, el qual relaciona cada nom amb un número d'identificació de classe. Mitjançant un editor de text, es crea un fitxer anomenat *labelmap.pbtxt* que es guarda en el directori \object_detection\training i s'escriu el següent:

```
item {
    id: 1
    name: 'Cadernera'
}
item {
    id: 2
    name: 'Pinsa borroner'
}
item {
    id: 3
    name: 'Passerell comu'
}
item {
```

```
14        id: 4
15        name: 'Pardal comu'
16    }
```

A continuació, es copia el fitxer de configuració del model a entrenar i s'enganxa en la carpeta \object_detection\training. Abans de començar l'entrenament s'ajusten certs paràmetres relacionats amb el model:

- S'indica el número de classes a entrenar, en aquest cas 4.
- S'indica el *path* als fitxers *train.record* i *test.record*.
- S'indica el *path* al fitxer *labelmap.pbtxt*
- S'indica el nombre d'imatges en el directori \images\test
- Es redueix el `batch_size` de 24 a 6 per evitar errors OOM (*Out of Memory*)

6.7.4. Entrenament.

Finalment ja es pot entrenar el model. Des del prompt d'Anaconda, s'activa l'entorn virtual i s'executa el fitxer *train.py* del directori \object_detection. Arribats a aquest punt, però, comencen a sorgir errors. Resulta que el codi descarregat del directori està realitzat per a versions 1.X de TensorFlow, i en les versions 2.X alguns mòduls s'han mogut i reanomenat. Així doncs, a partir de la guia de migració s'ha hagut d'anar buscant i modificant el codi per adaptar-lo a la versió 2.1 que és amb la qual s'està treballant. Tot i això, resulta que el mòdul *contrib,* que permet a la comunitat contribuir amb TensorFlow i garantir un manteniment d'aquestes contribucions, en les versions 2.X s'ha eliminat per complet.

Aquest fet ha forçat a generar un nou entorn d'Anaconda, anomenat *tensorflow2*, per tal d'instal·lar una versió anterior del programari. Es tria la versió 1.15 de TensorFlow i s'instal·len les corresponents versions de CUDA i cuDNN. Un cop preparat el nou entorn, es torna a executar el fitxer *train.py*, aquest cop sense errors.

Després d'uns minuts d'inicialització, el model es comença a entrenar. En aquest cas, no hi ha un nombre d'iteracions fixat, sinó que l'entrenament no s'atura fins que es premen les tecles "Ctrl+C" del teclat. Per aquest model en concret, SSD-MobileNet V2 Quantized Model, és recomanable entrenar-lo fins que la pèrdua es mantingui per sota de 2.

Per poder veure el progrés, s'executa TensorBoard en un nou prompt d'Anaconda. El que fa aquest mòdul és generar una pàgina web accessible des d'un cercador web on es mostren diverses gràfiques i informació sobre el progrés de l'entrenament. Una gràfica important és la de pèrdua, la qual mostra la pèrdua global del classificador al llarg del temps (Fig. 6.26).

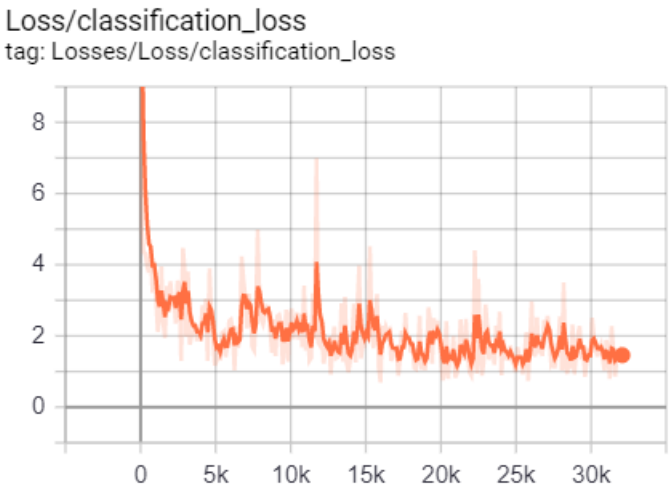


Fig. 6.26. Evolució de la pèrdua al llarg de l'entrenament.

Després de 9 hores d'entrenament i més de 30 mil iteracions, s'ha parat l'entrenament. En aquest punt el valor de la pèrdua està per sota del 2 i la gràfica no mostra pics pronunciats, la qual cosa apunta a que el valor es mantindrà per sota del límit establert. L'últim pas consisteix a exportar el model ja entrenat per poder utilitzar-lo. Es genera un fitxer .pb que conté el classificador de detecció d'objectes. Aquest serà el model que s'emprarà a la RPi per realitzar el reconeixement.

6.8. Implementació en Raspbian.

6.8.1. Conversió del model.

Per poder utilitzar en una Raspberry Pi el model exportat, primerament s'ha de convertir a un model TensorFlow Lite. Aquesta conversió es realitza mitjançant el Convertidor d'Optimització de TensorFlow Lite (TOCO, de l'anglès *TensorFlow Lite Optimizing Converter*) el qual requereix instal·lar TensorFlow des de la font, ja que certes instruccions no es poden emprar amb la versió preconstruïda de TensorFlow que s'ha estat utilitzant

fins ara. Se segueixen les instruccions oficials de la web de TensorFlow [14] que passen per instal·lar MSYS2 i Bazel.

Amb l'entorn preparat, ja es pot utilitzar TOCO per convertir el model. Des del prompt d'Anaconda s'activa Bazel i s'executa el model a través de l'eina de conversió, obtenint finalment un fitxer d'extensió .tflite. Resulta que Tensorflow Lite utilitza un format de mapa d'etiquetes diferent al de TensorFlow, així que es crea aquest nou mapa d'etiquetes i es guarda com a *labelmap.txt*.

```
Cadernera
Pinsa borroner
Passerell comu
Pardal comu
```

Un cop es tenen el model convertit i el mapa d'etiquetes, ja es pot utilitzar el conjunt per detectar objectes en imatges.

6.8.2. Preparació de l'entorn.

En cas que la Raspberry Pi sigui nova, primerament s'ha d'instal·lar el sistema operatiu. Per fer-ho es descarrega NOOBS (*New Out Of Box Software*), un assistent d'instal·lació de Raspbian, i s'extreuen els fitxers en una targeta microSD. Es col·loca la targeta a la placa i, en alimentar-la, apareix l'assistent d'instal·lació. Se segueixen els passos que es van indicant i un cop finalitzada la instal·lació es reinicia la RPi.

Donat que la placa no té perifèrics, per poder interactuar s'ha de disposar de teclat, ratolí i una pantalla HDMI. Per fer la interacció més còmoda, es procedeix a activar el VNC des del menú de preferències. Paral·lelament s'instal·la el software VNC Viewer al dispositiu des d'on es controlarà la Raspberry Pi, en aquest cas un ordinador. Arribats a aquest punt, ja està tot llest per començar a treballar.

Primerament es crea una carpeta, anomenada "tflite1", on es guardaran tots els fitxers necessaris per a l'execució de l'aplicació final. A continuació, s'instal·len les dependències necessàries per, posteriorment, instal·lar TensorFlow.

Seguidament, es passa el model *detect.tflite* i el mapa d'etiquetes *labelmap.txt* a un USB. Aquest s'introdueix a la placa i es copien ambdós fitxers en una nova carpeta dins el

directori de treball, /home/pi/tflite1/TFLite_model. En aquest punt ja està tot llest, únicament falta descarregar el codi que capta les imatges de la càmera i les passa al model per a realitzar el reconeixement, disponible a GitHub.

6.8.3. Resultats

Per tal d'avaluar el funcionament del detector, es procedeix a realitzar la detecció en imatges locals executant el codi *TFLite_detection_image.py*. Per assegurar que aquestes no formen part del *dataset* emprat durant l'entrenament, es busquen dibuixos realistes dels ocells (ja que el *dataset* està format íntegrament per fotografies).

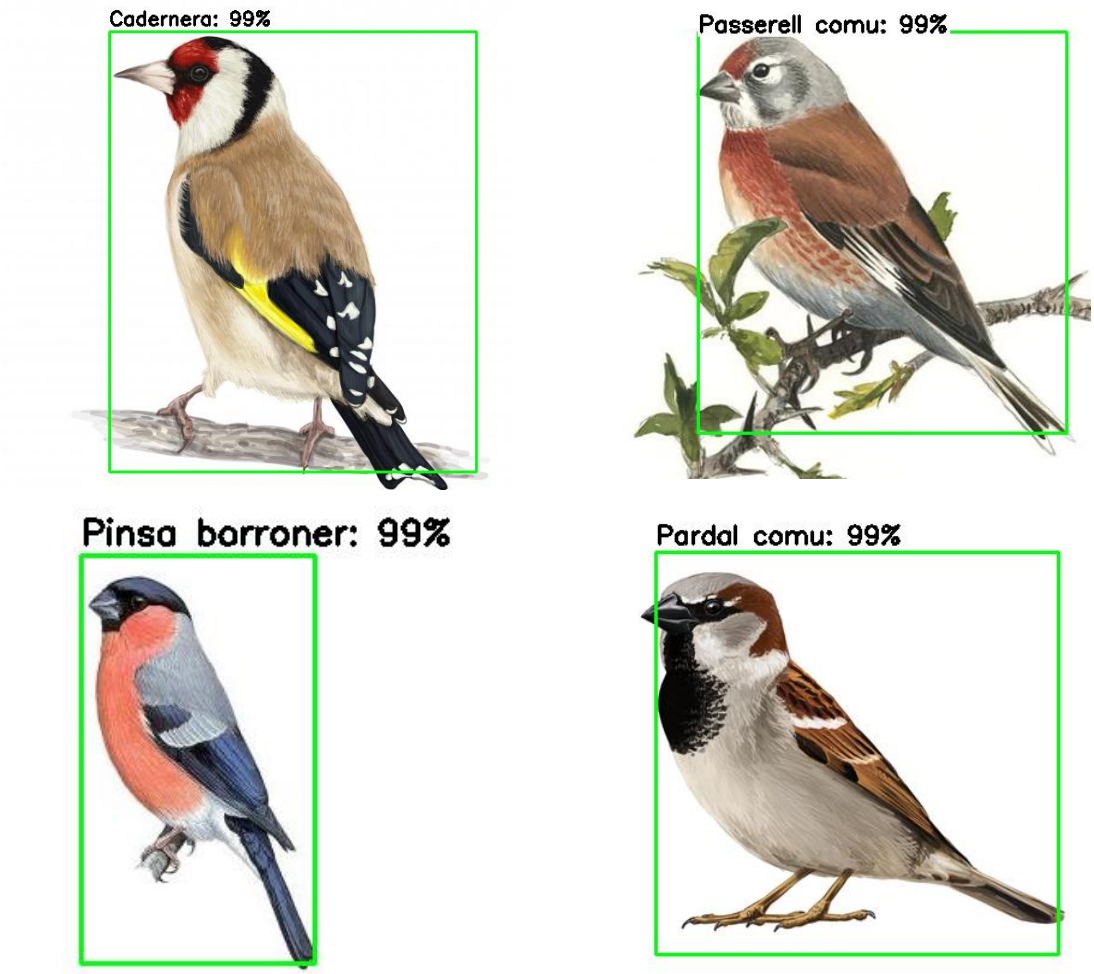


Fig. 6.27. Detecció d'objectes en imatges locals.

Els resultats obtinguts són excel·lents, el detector és capaç d'identificar correctament les quatre espècies d'ocells amb un 99% de fiabilitat.

A continuació, es realitza la mateixa prova mostrant les imatges a través de la càmera. Es comprova que estigui habilitada i s'executa el codi *TFLite_detection_webcam.py*. Després d'uns instants d'inicialització, s'obre una finestra on es mostren les imatges captades per la càmera.

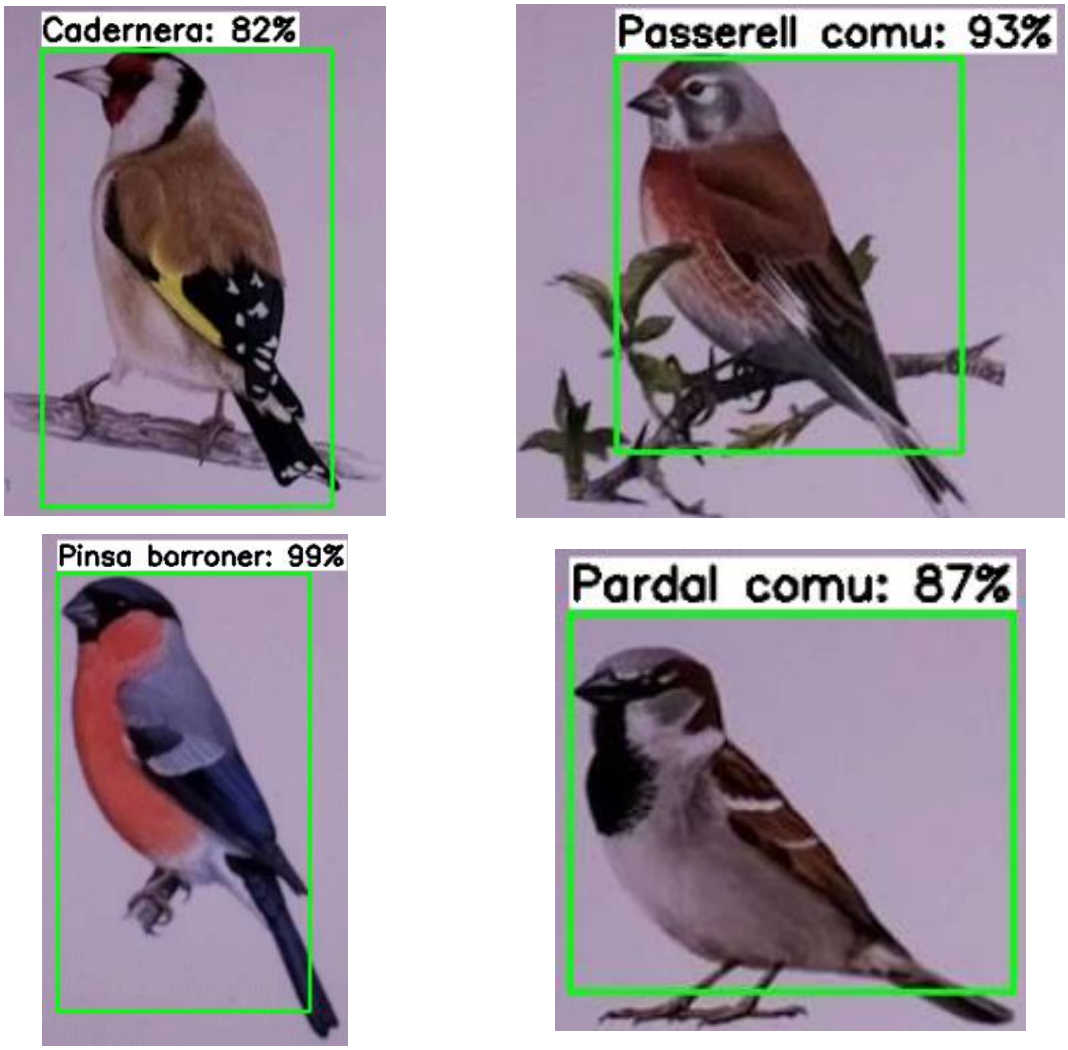


Fig. 6.28. Detecció d'objectes mitjançant la càmera.

En aquest cas, la probabilitat associada a cada espècie ha disminuït a excepció del pinsà borroner. Aquest fet pot ser degut a factors com la resolució de la imatge o la il·luminació. De totes maneres, les quatre espècies es reconeixen correctament i la probabilitat més baixa és del 82%, únicament un 18% inferior a l'obtinguda amb les imatges locals.

7. Captura selectiva d'ocells.

7.1. Interfície gràfica d'usuari.

Per tal de permetre a l'usuari triar quina espècie es vol atrapar, es dissenya una interfície gràfica d'usuari (GUI, de l'anglès *Graphic User Interface*). Aquesta es desenvolupa amb la llibreria *tkinter*, la qual està inclosa per defecte amb la instal·lació de Python.

La GUI (Fig. 7.1) està composta per quatre botons, corresponents a les quatre espècies, dels quals únicament es pot seleccionar un. Estan acompanyats del nom de l'ocell i d'una imatge per tal que sigui més visual i amigable per a l'usuari. Quan se selecciona una au, el nom de la tria s'emmagatzema en una variable anomenada `ocell`. A més a més, s'ha canviat el logo de la finestra pel de l'organització SEO/BirdLife.

També s'ha programat una funció `avis()`, la qual en ser cridada genera una finestra emergent amb un missatge tal com es mostra a la Fig. 7.2. El nom de l'ocell capturat, present en el missatge, varia en funció de l'espècie triada.

Fig. 7.1. Interfície gràfica d'usuari.

Fig. 7.2. Avís mostrat en realitzar una captura. Exemple de captura d'una Cadernera.

Cal mencionar que el codi s'ha desenvolupat inicialment en l'entorn Windows, i en passar-lo a la Raspberry Pi s'han hagut de realitzar algunes modificacions menors per tal que funcioni correctament.

7.2. Assemblatge i implementació.

Arribats a aquest punt, es disposa del model entrenat i exportat, del codi que importa aquest model i l'empra per a reconèixer imatges captades mitjançant una càmera i del codi referent a la interfície d'usuari. Així doncs, falta assemblar les diverses parts i afegir les línies de programa referents a les funcions dels dispositius electrònics.

L'estructura del programa (Fig. 7.3) es planteja en dos fitxers separats. El primer, anomenat *captura.py*, és el que s'inicia des del terminal. Un cop importades les llibreries i fitxers necessaris, es crida al segon codi, *interface.py*, el qual mostra la GUI. El programa espera fins que l'usuari selecciona una de les quatre opcions plantejades, i aleshores es guarda la tria en la variable `ocell`. Aquest valor se li passa al programa de captura i s'espera a que entri un ocell a la trampa, és a dir, s'activi el sensor FSR. En quant el sensor detecta presència, s'encén la càmera i s'inicia el reconeixement. En el moment que es detecta un ocell, es compara si és de la mateixa espècie que s'ha seleccionat prèviament. En cas que coincideixin, es comprova que el sensor no estigui activat i, tot seguit, s'activa l'actuador per tal de deixar caure la porta i tancar la trampa. Un cop s'ha atrapat l'ocell, es desactiva la càmera i es crida la funció d'avís del segon codi, el qual mostra el missatge per tal d'informar als ornitòlegs que han de recollir l'animal. En cas que, passats 2 minuts des de l'inici de la càmera no es detecti l'ocell seleccionat, es procedeix a apagar-la fins que es torni a activar el sensor.

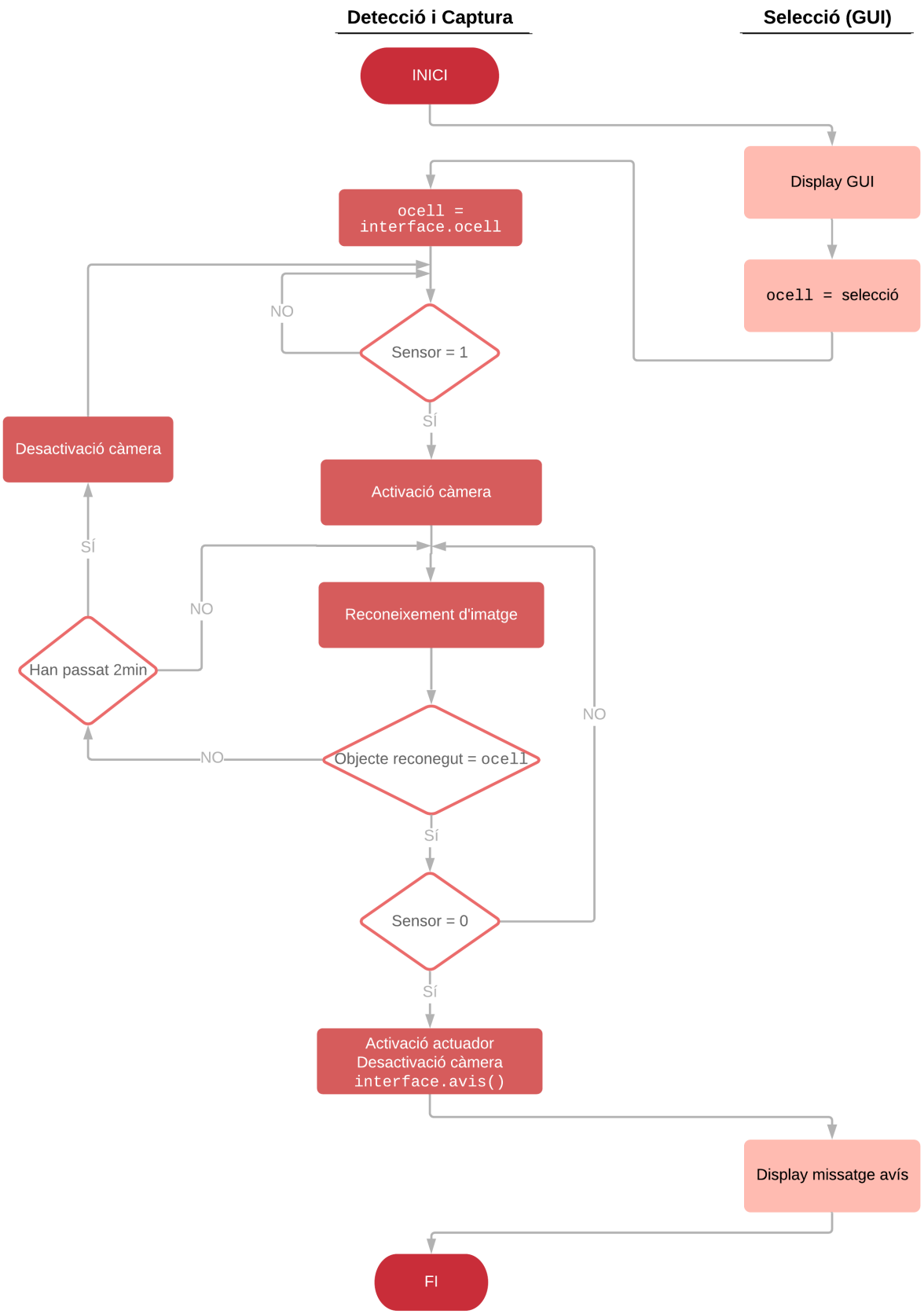


Fig. 7.3. Diagrama de flux del programa de captura.

Amb l'estructura definida en ment, es comença a programar. El primer pas consisteix a comunicar els dos fitxers. En el codi original de captura primerament es declara la classe `VideoStream`, en la qual es defineixen les funcions necessàries per iniciar la càmera, refrescar la imatge i parar-la. A continuació s'importen les llibreries de TensorFlow i el model. Finalment, s'inicia la càmera i s'entra en un bucle infinit, declarat mitjançant la condició `while TRUE`, on es realitza la identificació d'objectes en les imatges captades per la càmera.

En importar la GUI des del fitxer de captura sorgeix el primer problema. Resulta que, quan s'executa la línia d'importació, s'executa tot el codi de l'arxiu importat, de tal manera que es mostra la pantalla de selecció i s'entra en un bucle infinit del qual no es pot sortir. Aquest bucle està produït per la manera en què es programa la llibreria *tkinter* i no es pot evitar, ja que s'ha de crear un `mainloop()` per tal de mostrar la finestra. Està fet d'aquesta manera perquè la finestra s'ha de refrescar constantment per saber si l'usuari prem o no el botó. Per solucionar-ho, es crea una definició `run()` en la qual es posa tot el codi principal. D'aquesta manera, fins que no es crida la funció no s'executa el seu contingut.

A continuació, es treu la línia d'inicialització de la càmera. Aquesta s'activarà quan el sensor detecti que hi ha un ocell a la trampa. S'afegeix la crida de la funció `run()` abans de començar el bucle `while TRUE` i es guarda el valor de la selecció en la variable `ocell`. Aleshores apareix el segon problema.

Novament, quan es crida el codi per adquirir el valor de la selecció s'entra en el bucle, de tal manera que no es pot obtenir el valor fins que no es tanca la GUI. Aquesta vegada se soluciona forçant el tancament de la finestra, mitjançant la instrucció `destroy()`, un cop realitzada la selecció. Aquesta instrucció tanca únicament les finestres actives i no finalitza l'execució del programa.

Per verificar el correcte funcionament del codi, s'afegeixen instruccions `print()` que serveixen alhora de seguiment de l'aplicació i per informar a l'usuari de com avança l'execució del programa.

Un cop es comuniquen ambdós fitxers, s'afegeix la part del sensor i l'actuador. S'utilitza la llibreria *GPIO Zero* per interactuar amb els pins de la placa. Aquesta llibreria disposa de

diversos mòduls predefinits els quals compten amb funcions pròpies. Per a interactuar amb l'actuador s'empra el mòdul `LED`, amb el qual es pot activar i desactivar una sortida entre d'altres opcions. Quant al sensor, s'empra el mòdul `Button` que permet realitzar accions, per exemple, quan s'activa un senyal d'entrada, mentre està activat o esperar fins que deixa d'estar activat.

Per iniciar la detecció en quant s'activa el sensor, primerament es declara la variable `activar` i s'inicialitza a 0. Aleshores, s'afegeix la condició `if activar == 0` a l'inici del bucle `while TRUE` juntament amb un `else`. Com la variable s'inicialitza a 0, en entrar al bucle per primera vegada es compleix la condició del `if`, de tal manera que el programa es queda esperant a que s'activi el sensor i, aleshores, s'activa la càmera i la variable `activar` passa a valer 1. Tot el codi referent a la detecció es mou dins la condició `else`, de tal manera que s'executa quan ja s'ha detectat la presència d'un ocell.

A continuació, es cerca la variable que emmagatzema el nom de l'objecte detectat. Quan es localitza `object_name`, s'afegeix el codi per comparar aquest valor amb el de la selecció. Per evitar falses deteccions, es defineix la variable `deteccio` que, cada vegada que el valor de la selecció coincideix amb l'objecte detectat, incrementa el seu valor en 1. Aleshores, en cas que el seu valor sigui inferior a 2 s'afegeix un *delay* d'1 segon, de tal manera que una detecció ha de durar 2 segons abans activar l'actuador. D'altra banda, en cas que el sensor estigui detectant, es reinicia la variable `deteccio` a 0 per tal d'evitar que en tancar la porta es fereixi l'ocell o que aquest ja estigui fora quan s'activa la trampa.

Un cop han passat els 2 segons de detecció i no s'ha activat el sensor, s'activa l'actuador, es deixa caure la porta i es captura l'ocell. Seguidament es retornen les variables `activar` i `deteccio` a 0, s'apaga la càmera i es crida el missatge d'avís. En aquest punt es troba el següent error.

Quan es crida la funció `avis()`, al fons apareix la finestra principal en blanc (vegeu Fig. 7.4). L'objectiu és mostrar únicament el missatge, ja que l'altra finestra no conté cap informació. A més a més, aquest fet fa que el programa es quedi penjat: en prémer el botó OK en el missatge no passa res, aquest no desapareix i la finestra en blanc tampoc, i a part no es poden tancar cap de les dues finestres mitjançant la creueta.

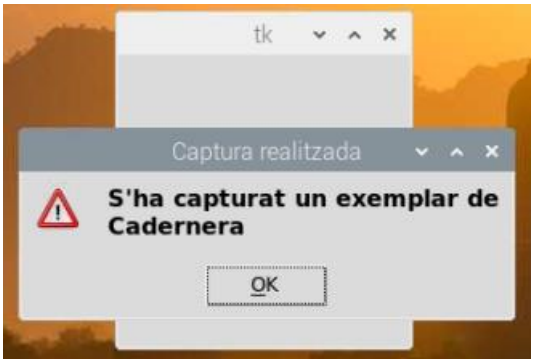


Fig. 7.4. Missatge d'avís i finestra principal en blanc.

Aleshores es busca com ocultar la finestra principal i es troba la funció `withdraw()`. Aquesta s'afegeix al principi de la funció `avis()`, però en executar-se es genera un error, de tal manera que no es mostra el missatge i el programa es finalitza. Per evitar aquest error es prova de posar el missatge d'avís en un fitxer a part. Aquesta solució funciona en Windows, però en la Raspberry Pi dona problemes, doncs en executar la GUI i realitzar la selecció, el valor no s'aconsegueix passar al codi principal i l'aplicació es queda penjada.

Finalment, després de donar-hi moltes voltes es decideix declarar el missatge en el fitxer de la GUI però dins d'una classe. Amb aquesta solució, l'aplicació permet ocultar la finestra de fons i importar el codi sense errors.

Per acabar, es realitza el codi per controlar el temps que està la càmera encesa. Quan s'activa la càmera, s'emmagatzema en la variable `t_ini` el nombre de cicles de rellotge realitzats des de l'inici de l'aplicació. Un cop finalitzat el reconeixement, es calcula el temps transcorregut des de l'activació de la càmera i, si és igual o superior a 2 minuts i no s'ha detectat l'espècie desitjada, s'apaga la càmera i es retorna la variable `activar` a 0.

Per verificar el funcionament del codi es redueix el temps a 10 segons, i en executar-lo sorgeix un nou problema. El primer cicle funciona correctament, però quan es reactiva la càmera es mostra l'últim frame captat abans d'apagar-la i no s'actualitza. Se soluciona editant la classe `VideoStream`. El codi original està pensat per finalitzar el programa en quant es para la detecció de la càmera. Per evitar-ho, s'elimina la línia de codi `self.stream.release()` de la funció `update()`, la qual tanca els recursos de la càmera.

7.3. Resultats.

Amb totes les parts de codi assemblades i funcionant, únicament queda provar l'aplicació resultant. En executar-la apareix un error referent al sistema de fitxers Hadoop. Aquest sistema no està instal·lat perquè no és necessari per a l'aplicació en qüestió. Independentment de si està o no instal·lat, TensorFlow sempre intenta importar-lo i, si no pot, emet un missatge d'error. Com s'ha mencionat, el sistema Hadoop no és necessari i no es fa servir en cap punt del codi, de tal manera que simplement s'ignora l'error i es comprova que l'aplicació funciona correctament.

Lamentablement, el rendiment no és l'esperat. Al cap de pocs minuts funcionant, el processador s'escalfa significativament, la qual cosa provoca que la velocitat de captura d'imatges de la càmera caigui d'1,24 fps a 0,3 fps. Aquesta baixada de rendiment és deguda al *Thermal Throttling*, un mecanisme de protecció que fa que un component, en aquest cas el processador, funcioni més lent per evitar danys per sobreescalfament.

Per evitar que s'activi aquest mecanisme, es compra un dissipador amb ventilador per refrigerar la placa. Un cop instal·lat, es comprova que la velocitat de captura d'imatges incrementa fins obtenir una mitja d'1,4 fps.

També es comprova que el valor dels fps varia en funció de com s'alimenta la placa. Inicialment es creu que la variació és deguda a la font d'alimentació, la qual varia entre un endoll, un port USB d'un ordinador i la bateria portàtil. Però després de realitzar diverses proves es veu que la caiguda de fps és deguda al cable micro USB emprat. En connectar el cable que venia inclòs amb la compra de la bateria portàtil s'obté una mitja de 0,7 fps, mentre que si es connecta un cable de millor qualitat, el rendiment és màxim amb una mitja d'1,4 fps independentment de la font d'alimentació emprada.

Tot i que 1,4 fps és un valor baix en comparació amb els 24 fps que captura, per exemple, una càmera de seguretat, aquests són suficients per desenvolupar l'aplicació en qüestió perquè l'esquer amb el qual s'atrau és menjar, de tal manera que l'ocell romandrà dins l'habitacle uns segons abans de marxar.

Finalment, es comprova satisfactòriament que la duració de la bateria és de 9h amb el màxim consum, és a dir, mantenint la càmera encesa independentment del sensor.

8. Impacte mediambiental.

Per a identificar l'impacte ambiental del projecte s'han realitzat llistes de control. A continuació es mostra una taula resum que s'ha confeccionat a partir de les llistes:

Accions impactants		**Conclusions**
Fase de construcció	Acústiques	Sorolls produïts en tallar la fusta. Impacte lleu.
	Visuals	Impacte nul.
	Residus	Trossos de la làmina de fusta sobrants. Impacte lleu.
Fase de funcionament	Acústiques	Impacte nul.
	Visuals	Espai ocupat per la trampa. Impacte lleu.
	Residus	Impacte nul.
Final de la vida útil	Acústiques	Impacte nul.
	Visuals	Impacte nul.
	Residus	Components electrònics com la placa, la càmera i la bateria entre d'altres. La resta d'elements com la fusta és reciclable. Impacte lleu.

Taula 8.1. Resum d'accions impactants durant les diferents fases del projecte.

Factor ambiental		Impacte
Medi natural	Atmosfera	Nul.
	Sòl	Nul.
	Aigua	Nul.
	Flora	Nul.
	Fauna	Ajuda a la seva conservació. Impacte positiu.
Medi socioeconòmic	Usos del territori	Nul.
	Cultural	Nul.
	Infraestructura	Nul.
	Humans	Nul.
	Economia i població	Nul.

Taula 8.2. Resum d'impacte en els factors ambientals.

Remarcar que el retorn d'aquest projecte és en matèria ambiental i no pas en l'àmbit econòmic. L'objectiu és protegir les espècies i col·laborar en la conservació dels seus hàbitats, mantenint així la biodiversitat del nostre planeta. Aquest projecte ho fa a petita escala, centrant-se en els ocells.

Un cop atrapada l'au, existeixen múltiples accions per a complir aquests objectius. Es pot dur a terme el marcatge en camp i alliberar l'animal en el mateix moment. Però també permet realitzar un posterior trasllat, ja sigui a un centre d'estudi per a prendre dades sobre el seu estat, a un centre mèdic per a realitzar controls de malalties o a un espai protegit per assegurar la supervivència de l'espècie, entre moltes d'altres opcions.

8.1. Anàlisi de riscos.

Un especialista en sensòrica comenta que es necessiten certs permisos de Protecció Civil per poder col·locar un objecte que contingui una bateria de liti a la natura, ja que aquesta es podria cremar i generar un incendi.

Per assegurar que no s'infringeix cap normativa i alhora informar-se sobre els permisos necessaris per desenvolupar el projecte, es procedeix a trucar a Protecció Civil. Quan s'aconsegueix contactar amb l'oficina, l'operador comenta que no hi ha cap normativa de Protecció Civil referent a què es pot o no col·locar a la natura, ja que aquest departament s'encarrega únicament de temes relacionats amb les persones.

L'operador també menciona que, en tot cas, si existeix alguna normativa referent al tema exposat aquesta formarà part del departament de Medi Ambient i recomana cercar la informació en la pàgina gencat.cat.

Així doncs, es procedeix a revisar la informació del departament de Medi Ambient present al web gencat.cat. En aquest, únicament hi ha normatives referents al reciclatge de les bateries de liti, però cap d'elles afecta a l'ús de la mateixa. De totes maneres, es contempla el risc d'incendi que suposa l'ús de la bateria i s'incorpora a les llistes de control mediambiental (vegeu Annex III).

9. Conclusions.

En el present projecte s'ha desenvolupat un mètode selectiu de captura d'ocells. S'ha realitzat un sistema de detecció d'objectes amb una Raspberry Pi i una Raspberry Pi Camera mitjançant l'ús de xarxes neuronals, juntament amb el disseny i creació d'una trampa així com la programació d'una interfície gràfica d'usuari.

La seva realització ha requerit l'aplicació de diversos coneixements:

- Modelatge 3D per al disseny de la trampa i la posterior generació de plànols.
- Electrònica i microcontroladors per al disseny dels circuits de l'actuador i el sensor i la seva connexió amb la RPi.
- Programació per al desenvolupament de la GUI, la programació dels components electrònics, el control de la càmera i la modificació dels fitxers de codi que realitzen la detecció d'objectes.
- Intel·ligència artificial i aprenentatge profund per al desenvolupament del model de detecció d'objectes i l'aplicació de *transfer learning*.

Tot i l'elevat nombre de problemes que s'han plantejat allarg del projecte, s'han pogut superar tots ells, obtenint finalment l'aplicació desitjada amb una elevada precisió de detecció. Cal mencionar que el rendiment no és òptim, ja que la velocitat de processament d'imatges és significativament lenta.

De totes maneres, els resultats globals són satisfactoris, ja que l'aplicació funciona correctament i és possible realitzar captures selectives d'aus. A més a més, la duració de la bateria és més que suficient per a poder realitzar diverses captures en un sol dia (9h quan el consum és màxim).

10. Futures línies de treball.

Es mostren els resultats finals a SEO/BirdLife i l'organització transmet la seva satisfacció en veure el funcionament del prototip. Tot i això, existeixen diverses millores que s'espera implementar en futurs projectes.

Primerament, en vista dels resultats obtinguts s'ha de valorar l'opció d'implementar el sistema de detecció d'objectes en una TPU, com pot ser Coral USB Accelerator de Google, tal com es va mencionar en l'avantprojecte. Aquest dispositiu està pensat i optimitzat per realitzar operacions d'inferència, de tal manera que l'execució del codi seria significativament més ràpida aconseguint un major nombre de fps.

També es plantejarà en futurs treballs l'increment de dades del *dataset*, millorar-ne la qualitat incloent imatges de major resolució i on l'ocell ocupi gran part de la imatge i augmentar el nombre d'espècies que reconeix model.

Quant al disseny, els circuits electrònics dels futurs prototips i dispositius s'implementaran mitjançant PCBs en lloc de *protoboards*. D'aquesta manera s'optimitzarà l'ús de l'espai, alhora que s'eliminen possibles problemes de desconnexions.

Finalment, a la trampa se li incorporaran LEDs de llum blanca per poder il·luminar-ne l'interior, de tal manera que es pugui realitzar el reconeixement en dies de poca llum i possibilitant l'ús de la trampa per a ocells nocturns.

11. Referències.

[1] www.birdlife.org, BirdLife International, *Who is BirdLife International?*

[2] www.seo.org, SEO/BirdLife, *Sobre nosotros*.

[3] Entrevista amb SEO/BirdLife. 15 de gener de 2020.

[4] www.ebird.org/spain/news/, mamadrid, *Nueva App eBird móvil 2.0 para Android - Tutorial*. 22 de setembre de 2019.

[5] www.merlin.allaboutbirds.org, The Cornell Lab, *Bird ID Wizard – Setp-by-step*.

[6] www.qz.com, M. Murphy, *This new Google-backed app will identify a bird just from a photo*. 9 de juny de 2015.

[7] www.seo.org, SEO/BirdLife, *A partir de ahora no podrás decir que no conoces un pájaro*. 27 de desembre de 2016.

[8] www.elmundo.es/ciencia, M.G. Corral, *Una nueva 'app' permite identificar más de 550 aves desde el teléfono*. 15 de febrer de 2014.

[9] www.educa2.madrid.org/web/miguelangel.diaz1, M.A. Díaz, *Captura ilegal de aves silvestres*.

[10] www.computerhoy.com, J.A. Pascual, *Inteligencia artificial: qué es, cómo funciona y para qué se está utilizando*. 24 d'agost de 2019.

[11] www.coursera.org, A. Ng, *Redes neurales y aprendizaje profundo*.

[12] MathWorks, "Introducing Deep Learning with MATLAB". Versió digital.

[13] www.github.com/tensorflow/models, *TensorFlow Model Garden*.

[14] www.tensorflow.org, TensorFlow, *Build from source on Windows*.

[15] www.seo.org/quienes-somos/seobirdlife-en-numeros/, SEO/BirdLife, *En Números*.