Treball Fi de Grau

Grau en Enginyeria en Tecnologies Industrials (GETI)

Sistema d'identificació i regressió del pes en ous de gallina basat en Deep Learning

MEMÒRIA

Autor: Jaume Mora Juvanteny

Director: Lluís Solano Albajes

Convocatòria: Febrer 2020

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

Resum

En un període de temps relativament curt, el *Deep Learning* ha fet un gran progrés en el camp de la visió per computadors, obtenint resultats increïbles. Així ho demostren algunes de les seves aplicacions com el reconeixement facial, la conducció autònoma de cotxes o la detecció i classificació d'objectes, que ja s'han convertit en part del nostre dia a dia.

En aquest treball es pretén desenvolupar un sistema capaç de localitzar ous de gallina en imatges i predir-ne el pes. Per fer-ho, s'entrenarà un model neuronal i per tal que l'usuari pugui fer prediccions i visualitzar els resultats obtinguts d'una manera més agradable, es crearà un aplicatiu web.

Al llarg d'aquesta memòria, el lector podrà veure totes les etapes que inclou un projecte de *Deep Learning*: la creació del *dataset*, el processament de les dades, el desenvolupament i configuració del model, l'entrenament i finalment l'avaluació. El treball també inclou les eines utilitzades, explicacions sobre els conceptes bàsics d'aquest camp i una descripció detallada de la solució implementada.

Tant la redacció del projecte com el codi adjunt estan pensats per a no requerir de grans coneixements de programació ni de *Deep Learning*, de manera que s'intentarà que les explicacions siguin el més planeres possibles.

Sumari

Índex de figures

Índex de taules

1. Glossari

Sigles i acrònims:

CNN	Convolutional Neural Network
DL	Deep Learning
DNN	Deep Neural Network
PNG	Portable Network Graphics
XML	Extensible Markup Language
CSV	Comma-separated value
CSS	Cascading Style Sheets
HTML	Hypertext Markup Language
IoU	Intersection Over Union
ReLU	Rectified Linear Unit
MSE	Mean Squared Error
SGD	Stochastic Gradient Descent
RGB	Red Green Blue

2. Prefaci

2.1 Origen i motivació del projecte

L'origen del projecte neix de l'interès per voler aplicar visió per computadors a un procés manual repetitiu com podria ser el de mesurar l'alçada d'un objecte, comprovar el codi d'una etiqueta o, com és el cas d'estudi, pesar un producte. En l'actualitat, en plena era de la indústria 4.0, cada cop són més les empreses que estan incorporant aquestes tècniques als seus processos per tal de ser més eficients i competitius. Així doncs, aquest projecte pretén ser una simulació (a petita escala) d'aquests processos. A més, volia que el treball no fos únicament un estudi teòric, sinó que tingués una part pràctica de disseny i implementació d'una proposta de solució.

Inicialment es volia dissenyar un sistema que, també mitjançant la visió artificial, estimés el pes d'animals. Posteriorment, però, es va considerar que l'obtenció de les dades (imatges i els pesos d'aquests) podia suposar un problema, de manera que es va acabar optant per fer la predicció del pes d'ous de gallina, un producte molt més fàcil d'adquirir.

Per altra banda, aquest treball també el volia plantejar com un repte personal. Hi ha una demanda creixent de perfils professionals especialitzats en intel·ligència artificial i *data mining*, i saber interpretar dades i trobar solucions òptimes és un requisit en tots ells. És per això que volia aprofitar aquest projecte per tenir una primera presa de contacte amb aquests sectors emergents.

2.2 Anglicismes

En la redacció d'aquesta memòria s'han utilitzat molts anglicismes pel fet que en la majoria d'estudis en el camp de la intel·ligència artificial són en anglès, i sovint no existeix una traducció consensuada o una paraula equivalent en català.

En la meva opinió, penso que pel bé de la comunitat científica és millor mantenir els conceptes en anglès, ja que és una manera d'estandarditzar aquest camp i evitar confusions a l'hora de traduir aquestes paraules.

3. Introducció

3.1 Objectius del projecte

L'objectiu principal del projecte és crear una eina capaç de processar i analitzar imatges d'ous de gallina i predir-ne el pes amb el mínim error possible, agilitzant d'aquesta manera el procés de pesatge tradicional i demostrant que, mitjançant el *Deep Learning*, es poden aconseguir bons resultats.

Tot i que en un principi es va plantejar el problema com una tasca de classificació[1], finalment s'ha encarat com una regressió del pes, ja que s'ha considerat un repte més interessant. L'estimació del pes de productes és una problemàtica comuna en moltes aplicacions del món; hi ha molts mètodes per calcular el volum d'objectes, però la majoria d'ells estan basats en imatges 3D (cosa que dificulta obtenir resultats en temps real i els equipaments per obtenir aquestes imatges 3D són molt cars). És per això que aquest treball se centrarà en imatges 2D.

Un altre objectiu és el d'evitar els possibles errors humans durant fase de pesatge: un procés que necessita a una persona pesant els ous i anotant els seus pesos de manera repetitiva, sempre és més susceptible a errors en comparació de si ho fa un sistema automàtic. En l'estudi [1] es demostra el que s'ha comentat.

Com a darrer objectiu, es vol crear una interfície gràfica (en forma d'aplicació web) per tal de fer prediccions i visualitzar els resultats d'una manera agradable. Això permet que, encara que l'usuari no tingui coneixements en *Deep Learning*, sigui capaç d'utilitzar el sistema sense cap mena de problema, ja que es dissenyarà per tal que sigui el més intuïtiu possible.

[1] Els ous de gallina es poden classificar en 4 talles possibles en funció del seu pes: talla S (inferior a 53 g), talla M (entre 53 g i 63 g), talla L (entre 63 g i 73 g) i talla XL (superior a 73g)

3.2 Abast del projecte

El propòsit d'aquest treball és que sigui un prototip inicial i únicament permeti fer-se una idea del potencial que té la visió per computadors i el *Deep Learning* en aplicacions que podrien ser perfectament reals.

4. Anàlisi del problema

4.1 Treballs relacionats

Per entendre millor el cas d'estudi, el primer que es va fer va ser buscar treballs relacionats amb aquest problema i veure quines solucions proposaven i quines conclusions en treien. A continuació es farà un resum dels que s'han considerat més destacats:

4.1.1 Object weight estimation from 2D images

El sistema proposat en aquest treball [2] utilitza únicament el processament d'imatges 2D per calcular el pes d'objectes. En aquest estudi simplement es pren una imatge d'entrada, s'identifiquen i classifiquen els objectes que hi apareixen mitjançant segmentació[2], es mesura el volum de cada objecte i es calcula el pes a partir del volum mesurat i la densitat de l'objecte (emmagatzemada en una base de dades del sistema).

Pel càlcul del volum, a més de l'àrea de l'objecte (extreta a partir dels píxels de la imatge), es necessiten les mides reals d'aquest i per aconseguir-les es fa a través d'un objecte de referència, en aquest cas el dit de l'usuari.

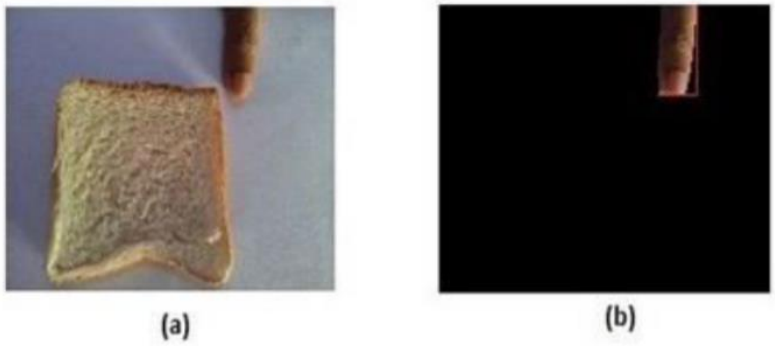

Figura 4.1. (a) Imatge de test amb dit i (b) càlcul de les dimensions del dit, extreta de [2]

[2] En el camp de la visió per computadors, la segmentació és el procés de dividir una imatge digital en diverses parts (grups de píxels) per posteriorment identificar-les i saber què representa cada una d'aquestes parts.

Inicialment es fa un calibratge on es captura la imatge d'un dit i se'n guarden les seves mides reals. Cada vegada que es fa una foto amb diversos objectes a analitzar, es col·loca el dit al costat, de manera que el sistema pot obtenir les dimensions reals d'aquests. Posteriorment es fan els càlculs necessaris per trobar el volum i, finalment, el pes de cada objecte.

En les conclusions s'explica que amb aquest sistema s'aconsegueixen resultats amb un 97% de precisió.

4.1.2 Weight estimation through Image Analysis

Aquest estudi [3] es basa en capturar imatges amb objectes en un fons fix (de dimensions conegudes), trobar les dimensions reals de l'objecte i a partir d'aquí calcular el volum i el pes.

En aquest cas, per l'obtenció de les dimensions reals de l'objecte s'utilitza una tècnica de proporcionalitat: a partir de les dimensions reals del fons i de les dimensions (en píxels) del fons però dins la imatge un cop processada, es pot calcular una proporció d'escala (*scaling ratio*) que ens servirà per trobar les mides reals dels objectes de la imatge.

En l'esquema següent es pot veure el procediment seguit en aquest treball:

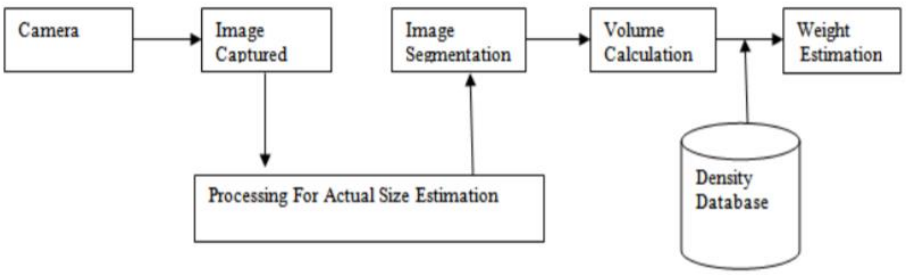

Figura 4.2. Esquema dels passos seguits per a l'obtenció del pes en l'estudi [3]

Dues vistes perpendiculars (una frontal i una lateral) han estat utilitzades per obtenir l'estimació del volum de cada objecte, que multiplicat per la densitat permet obtenir el pes.

Així doncs, amb aquest mètode es poden obtenir resultats d'aproximadament un 90% de precisió.

4.1.3 Classification of Similar Objects of Different Sizes Using a Reference Object by Means of Convolutional Neural Networks

En aquest treball [4] es busca classificar cargols d'un mateix model, de manera que on l'única diferència entre ells són les dimensions (es consideren 8 tipus de cargols, de M2 a M8).

Per fer-ho, s'utilitzen tècniques de *Deep Learning*, concretament les que es coneixen com a xarxes neuronals convolucionals (*Convolutional Neural Networks* en anglès, o CNN). Pel que fa al disseny de l'estudi, es considera sempre la mateixa distància fixa, de manera que els cargols més grans es mostren més grans en les imatges.

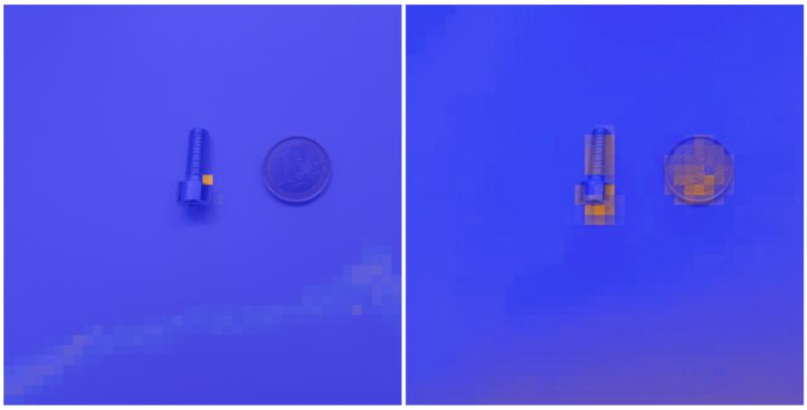

Figura 4.3. A l'esquerra veiem el mapa de calor a l'inici de l'entrenament, mentre a la dreta veiem el mapa de calor un cop ha acabat (utilitza la informació de la moneda). Referència [4]

En aquest estudi es demostra que les CNN són capaces de mesurar implícitament els cargols de les imatges utilitzant objectes de referència (en aquest cas una moneda d'un euro) i integrar-los en el procés d'aprenentatge, tal com es veu en la figura anterior.

Utilitzant imatges de cargols amb una referència s'incrementa la precisió en un 40%, tal i com es pot observar en la taula següent:

Reference Object	Identification accuracy
Without	57.5 %
With	**97.5** %

Taula 4.1. Comparativa de la precisió amb o sense objecte de referència, extreta de [4]

4.2 Disseny de l'estudi

Un cop llegits i estudiats els treballs anteriors, es dissenya l'estudi tenint en compte les següents consideracions:

- En el procés de captura d'imatges dels ous de gallina, la distància entre aquests i la càmera serà sempre fixa.
- En tota imatge hi haurà sempre un únic ou.
- Es busca que el model sigui capaç d'identificar l'ou i trobar-ne el pes independentment de la posició d'aquest; per tant es faran fotos dels ous amb diferents orientacions (dret del tot, tombat o una mica inclinat)
- Es compararan els resultats obtinguts amb un model entrenat amb objectes de referència respecte a un model entrenat únicament amb imatges d'ous sols (sense objecte de referència).
- Per tal de treballar amb una quantitat més gran d'imatges, es faran diverses fotos de cada ou, però fent-hi petits canvis (així semblarà que es disposa de més imatges).

Ara que ja es tenen establertes les bases de l'estudi, anem a veure les diferents etapes que s'han seguit per la realització del projecte.

5. Dataset

En aquest apartat es farà una breu explicació de tot el que fa referència al *dataset* del projecte; un *dataset* no deixa de ser res més que el conjunt de dades, ordenades i tabulades, del problema d'estudi en qüestió. En qualsevol projecte de *Deep Learning* es requereix una gran quantitat de dades per tal de poder entrenar els models i que aquests siguin capaços de reconèixer patrons o detectar anomalies a l'hora de fer prediccions.

Avui en dia, hi ha molts grups d'investigació, empreses i institucions que han alliberat les seves dades de manera que qualsevol persona les pot descarregar i utilitzar lliurement en els seus projectes. Alguns exemples de *dataset* serien:

- **MNIST:** conté 70.000 imatges de dígits escrits a mà amb les seves corresponents etiquetes. Està molt bé per aprendre els conceptes bàsics i entrenar models en el reconeixement de patrons.
- **COCO:** conté 330.000 imatges, amb 80 categories d'objectes diferents. És molt popular i està molt utilitzat en tasques de segmentació i detecció d'objectes.
- **ImageNet:** és un *dataset* que conté aproximadament 1.500.000 imatges. Cada imatge conté una frase que descriu el contingut de la imatge. És molt utilitzat en projectes de detecció i reconeixement d'objectes.

En aquest projecte, en voler-se dur a terme una tasca tan específica com és la del reconeixement d'ous de gallina i la predicció del pes, es va optar per crear un dataset nou, ja que no n'hi havia cap de públic amb aquesta informació.

5.1 Creació del dataset

Abans de començar a crear el *dataset*, és fonamental tenir clar què és el que es vol que el model sigui capaç de fer, i per tant la informació que se li ha de proporcionar per tal que pugui dur a terme aquesta tasca. Com ja hem comentat abans, el model ha de localitzar l'ou en una imatge i seguidament predir-ne el pes.

Així doncs, cal proporcionar la informació de:

- **Localització de l'ou (*bounding box*):** rectangle en el pla 2D que emmarca l'ou sobre la imatge. Ve descrit per un vector de 4 coordenades, (x, y, w, h), on x i y

són les coordenades del punt inferior esquerre, i *w* i *h* són l'amplada i l'alçada respectivament de la *bounding box* (requadre que l'emmarca).

- **Pes de l'ou:** variable contínua que representa el pes de l'ou (en grams).

El procés de creació del dataset que es va seguir consta de les següent parts:

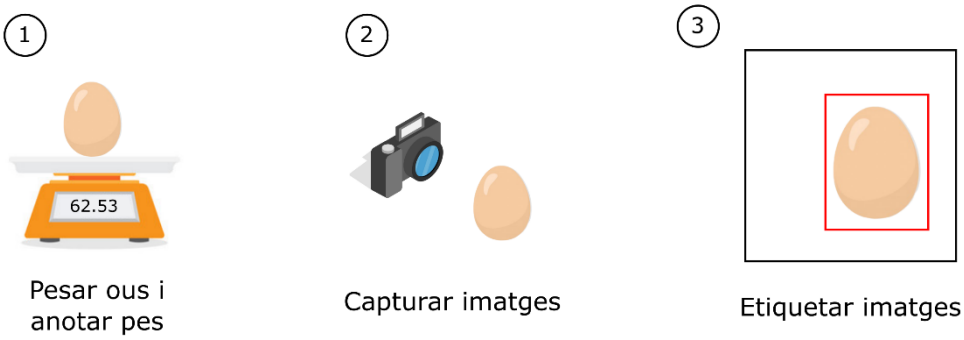

Figura 5.1. Esquema de les fases que conformen la creació del dataset

5.1.1 Pesar ous i anotar pes

El primer pas que es va seguir va ser el de pesar els ous i anotar-ne el pes en una taula d'Excel. Es va fer servir una balança digital amb 2 decimals de precisió. Per tal que no hi hagués discrepàncies entre els diferents ous, es van pesar sempre de la mateixa manera (posició horitzontal). Un cop estabilitzat el resultat de la balança, s'entra en una taula d'Excel (que posteriorment es convertirà a CSV per tal de tractar-ho amb Python). El format era el següent:

Id	Nom imatge	Pes de l'ou (g)	Talla
1	DSC02092020.jpg	62,53	M

Taula 5.1. Dades obtingudes després de pesar un ou

5.1.2 Capturar imatge

Com s'ha comentat anteriorment, els *datasets* solen contenir una quantitat enorme de dades (de la mida de 10.000 imatges). Evidentment comprar aquesta quantitat d'ous suposa una despesa molt gran, de manera que es va optar per fer més d'una foto a cada ou (concretament 5 fotos per ou) afegint petites variacions en cada una, com ara canvis

de posició de l'ou o canvis en la llum de l'ambient, i així simular que es disposaven de més ous i aconseguir que la mida del *dataset* fos major.

Encara que sembli que aquests canvis puguin alterar el model i que a l'hora de predir els resultats no funcioni tan bé, en realitat el que fan és ajudar al model a generalitzar[3]. Així doncs, un cop entrenem el model amb aquestes imatges, entendrà que encara que l'ou estigui tombat enlloc de vertical, o que hi hagi menys llum a l'ambient, o que l'angle de la foto hagi variat lleugerament, el que està veient segueix sent un ou.

Per tant, la taula anterior es va modificar de la següent manera.

Id	Nom imatge	Pes de l'ou (g)	Talla
1	DSC020920201.jpg	62,53	M
1	DSC020920202.jpg	62,53	M
1	DSC020920203.jpg	62,53	M
1	DSC020920204.jpg	62,53	M
2	DSC020920205.jpg	62,53	M
2	DSC020920206.jpg	67,81	L
2	DSC020920207.jpg	67,81	L

Taula 5.2. Exemple d'informació emmagatzemada després de pesar diversos ous

L'identificador indica l'ou real, el nom de la imatge és un de diferent cada cop, i les dades de pes i talla es mantenen amb l'identificador, ja que representen diferents fotos però del mateix ou (per tant la talla i el pes són els mateixos, sempre que l'ou sigui el mateix). Es va posar la talla com a informació addicional per si es més endavant es volia fer alguna prova de classificació.

[3] En el món del *Deep Learning* el concepte de generalitzar es fa servir per referir-se que el model és capaç d'extrapolar el que ha après durant l'entrenament a altres dades que no ha vist mai. Quan un model no generalitza vol dir que només troba bé les solucions en l'entrenament.

5.1.3 Etiquetatge de les imatges

Finalment, un cop fotografiats tots els ous cal etiquetar-los un per un per tal de marcar on es troba l'ou en cada imatge.

Per fer-ho es va utilitzar un programa anomenat labelImg [5]. Aquesta és una eina gràfica per anotar imatges molt utilitzada en els projectes de *Deep Learning*. El seu funcionament és molt simple: únicament s'ha d'obrir la carpeta on es troben les imatges, seleccionar la primera i anar marcant amb un rectangle la localització de l'ou.

Després d'etiquetar aquella imatge, es clica "Guardar" i automàticament es carrega la següent imatge del *dataset*. Les etiquetes que contenen la informació de la localització de l'ou es guarden com a XML en format PASCAL VOC (format utilitzat en el *dataset* ImageNet).

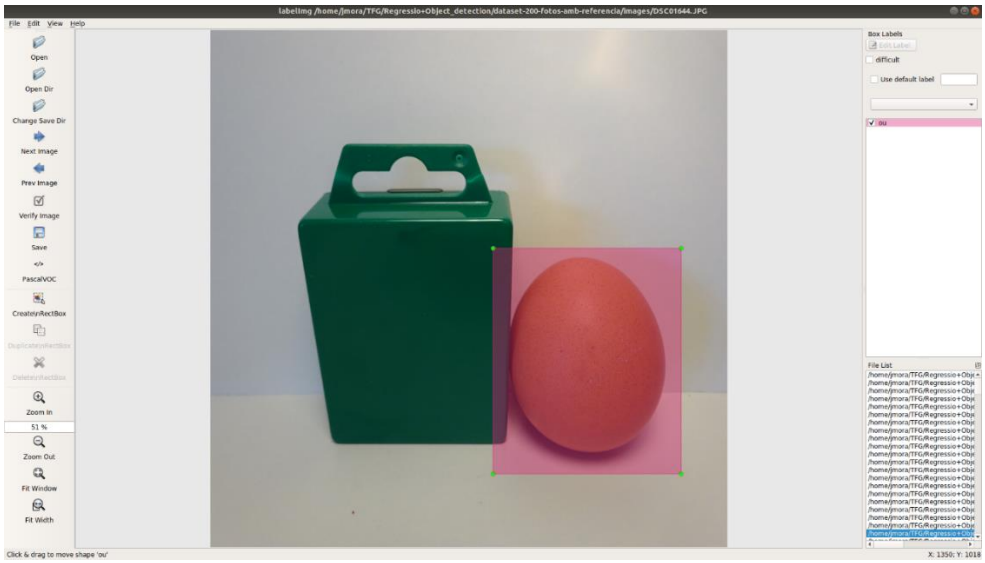

Figura 5.2. Captura de pantalla del labelImg

5.2 Tractament de dades

Un cop creat el *dataset*, per tal d'ordenar les dades es decideix crear un sol fitxer que contingui tota la informació junta (nom de la imatge, *bounding box*, pes i talla). D'aquesta manera, es van unir els fitxers XML individuals de cada imatge en un CSV conjunt, i s'hi va afegir el pes i la talla corresponents a cada imatge (utilitzant llibreria de pandas [6] de Python).

El fet de guardar les mides de la imatge quan es va etiquetar permet es redimensionar la *bounding box* a la mida que es vulgui (dividint les coordenades de la bbox per les mides de la imatge original, i llavors multiplicant per les noves mides). mida

Això dona molta flexibilitat a l'hora de poder manipular les imatges i poder donar-los-hi la mida que es vulgui, sense perdre mai la localització de l'ou ni d'haver d'etiquetar altre cop tot el *dataset* de nou.

El fitxer final va quedar de l'estil següent:

Nom imatge	Amplada original	Alçada original	x bbox	y bbox	w bbox	h bbox	Pes	Talla
DSC0209201.jpg	1920	1920	60	60	80	80	65.3	M

Taula 5.3. Fitxer final amb les dades de la localització i del pes de cada ou

D'aquesta manera, la informació ja queda preparada per quan entrenem el nostre model. Cada cop que carreguem una imatge, tindrem tota la informació corresponent a aquesta en cada fila de la taula (*dataframe*[4]).

5.3 Dades d'entrenament, validació i test

Per entrenar un model de *Deep Learning* i saber si funciona correctament cal dividir el conjunt de dades disponibles en dos grups: dades d'entrenament (*training*) i dades de prova (*test*). En general aquests grups es divideix en una proporció 80-20 (80% d'entrenament i 20% de prova), i s'agafen les mostres de manera aleatòria (no en seqüència sinó barrejant les dades). Al mateix temps, una part de les dades d'entrenament es reserva com a dades de validació (*validation*).

Les dades d'entrenament que ens queden després de treure les de validació i de prova són les que s'utilitzen per "alimentar" l'algoritme, mentre que les de validació s'utilitzen per ajustar la configuració del model i fer un seguiment del procés d'entrenament (ens guien per veure si anem pel bon camí).

[4] En la ciència de dades es coneix com a *dataframe* una estructura de dades ordenada i tabulada. Presenten una estructura molt similar a una matriu, però a diferència d'aquesta, un

És important notar que quan ens esforcem per millorar l'algoritme ajustant els paràmetres gràcies al comportament del model amb les dades de validació, estem incidint en el model "indirectament", que pot esbiaixar els resultats a favor del conjunt de validació. D'aquí la importància de disposar d'un conjunt de dades de test reservat per una prova final, amb les dades que el model no ha vist mai abans durant l'etapa d'entrenament (ni com a dades d'entrenament ni com a dades de validació). Això permet obtenir una mesura de comportament de l'algoritme més objectiva i avaluar si el nostre model generalitza correctament.

És important recalcar que les dades de test les utilitzem per fer prediccions: això significa que el model no s'entrena ni incorpora coneixement amb aquestes dades, simplement es limita a veure una entrada i treure una sortida. Posteriorment, amb aquesta sortida predita i amb el valor vertader de sortida (que tenim guardat) calculem l'error comès pel model. Els resultats que ens podem trobar són:

- Si la precisió en test és propera a la d'entrenament significa que el nostre model entrenat està generalitzant bé i el podem donar-lo per bo.
- Si la precisió en test és molt diferent de la d'entrenament, tant per sobre com per sota, llavors és un indicador que el nostre model no ha entrenat bé i no ens serveix. De fet podria ser un cas de *overfitting*[5].

La figura següent és un esquema visual del repartiment de les dades explicat, i de la funció de cada conjunt:

[5] En el camp del *Deep Learning* parlem d'*overfitting* (o sobreentrenament) quan el model obtingut s'ajusta tant als exemples etiquetats de l'entrenament que no és capaç de realitzar les prediccions correctes amb dades noves que no ha vist mai. És a dir, el model aprèn detalls que no són generals.

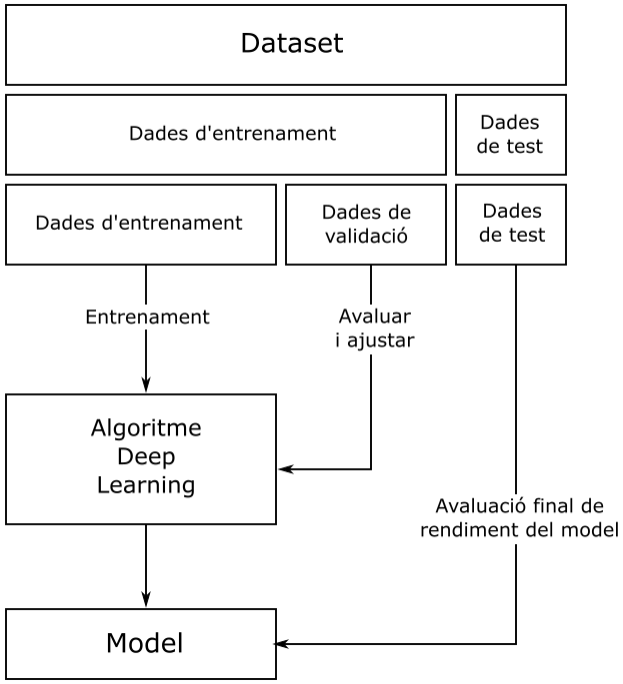

Figura 5.3. Repartiment de dades en tres grups (entrenament, validació i prova), extreta de [5]

Així doncs, pel nostre estudi es van crear dos *datasets*:

- *Dataset* amb 300 fotos d'ous amb referència (caixa verda). Aquest *dataset* va ser creat a partir de 60 ous reals (5 fotos per ou).
- Un altre *dataset* de 300 imatges també, però a diferència de l'anterior sense referència. Va ser creat a partir de 60 ous reals (els mateixos que en l'altre cas).

6. Xarxes neuronals convolucionals

6.1 Introducció a les CNN

En *Deep Learning*, una xarxa neuronal convolucional (en anglès *Convolutional Neural Networks o CNN*) és un tipus de xarxa molt utilitzada en l'anàlisi visual d'imatges. Durant els darrers anys s'ha anat popularitzant molt a l'aconseguir uns resultats increïbles en el camp de la visió per computadors.

Aquestes xarxes convolucionals són molt semblants a les xarxes neuronals tradicionals (o densament connectades): també estan formades per neurones, amb els seus respectius paràmetres que es van modificant a mesura que es va "entrenant la xarxa". Ara bé, un dels trets diferencials d'aquests tipus de xarxes és que assumeixen que l'entrada és una imatge, fent així el procés d'aprenentatge més eficient.

A continuació s'explicarà per sobre el funcionament de les CNN, però perquè ens en fem una petita idea, aquestes xarxes imiten el comportament de l'ull humà, i d'aquesta manera aconsegueixen "veure" i "reconèixer" objectes tal i com fem nosaltres. Les primeres capes són les encarregades de detectar coses simples com línies, formes o textures, i a mesura que ens anem endinsant a la xarxa, les capes del final són capaces de reconèixer coses més complexes com ara un animal o una cara.

L'exemple següent il·lustra aquest funcionament:

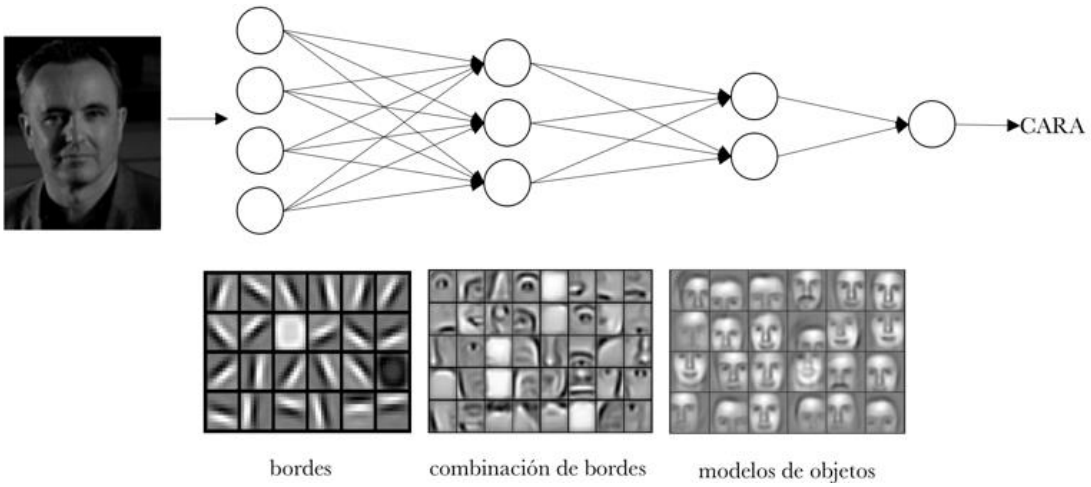

Figura 6.1. La primera capa aprèn elements bàsics, la segona aprèn patrons compostos i així successivament fins a aconseguir reconèixer cares. Imatge extreta de [5]

6.2 Arquitectura de la xarxa

En aquesta secció ens fixarem en l'estructura de la xarxa neuronal del projecte i s'intentarà explicar breument com funciona i quines són les parts més importants. A simple vista, veiem que la xarxa està formada per una branca principal i té dues sortides, que seran les que faran la predicció de la *bounding box* i del pes.

L'estructura de la xarxa neuronal convolucional del projecte, esquemàticament la podríem representar de la següent manera:

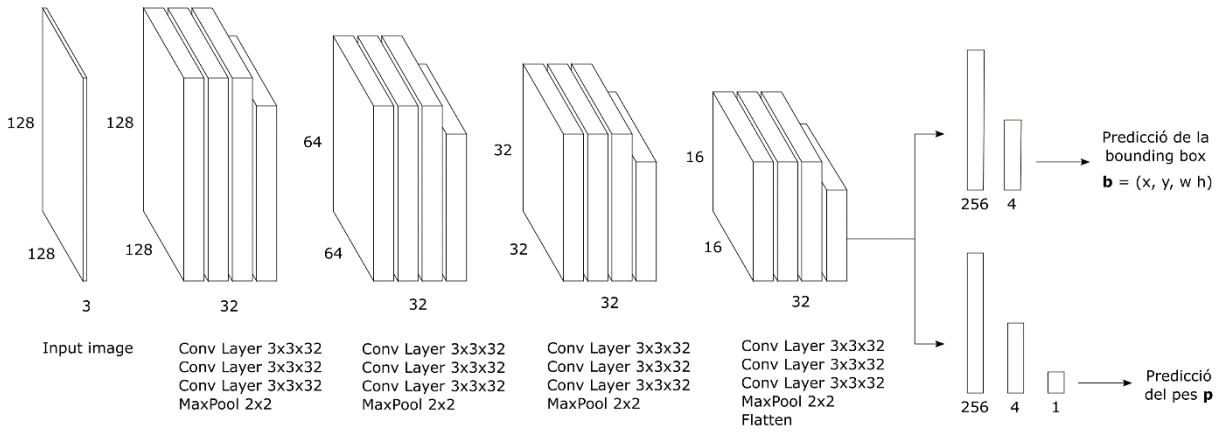

Figura 6.2. Esquema de la xarxa neuronal convolucional del projecte

La xarxa es pot dividir en tres parts: entrada, capes ocultes (blocs intermedis) i sortides. A continuació s'explicarà cadascuna de les parts.

6.2.1 Entrada i normalització de les dades

La xarxa pren com a entrada una imatge, que tractarem com un tensor 3D amb dos eixos d'alçada i amplada, i un tercer eix de profunditat (també anomenat eix de canals). Així doncs, en el cas de les imatges en escales de grisos, tan sols tenim un color i per tant la profunditat serà 1, mentre que en el cas de les imatges en color RGB tindran una profunditat 3, ja que la imatge té tres canals: vermell, verd i blau.

Abans de començar a introduir imatges a la xarxa, convé normalitzar els valors dels píxels d'aquestes. Els píxels tenen valors que van de 0 a 255, de manera que dividirem cada píxel entre 255 i ens quedarà un valor entre 0 i 1 (no entrarem en detall de per què es fa això, però és una pràctica comuna a l'hora d'entrenar xarxes neuronals).

La mida de les imatges d'entrada és un paràmetre crític ja que imatges molt grans permeten detectar amb claredat detalls molt petits, però per altra banda fan que el procés d'entrenament sigui molt més lent, ja que hi ha molta més informació a manipular. És per això que s'ha escollit una mida d'imatges d'entrada de 128x128 píxels.

També s'han considerat sempre les imatges quadrades (en cas que no ho sigui es farà un redimensionament per tal de convertir-la en quadrada).

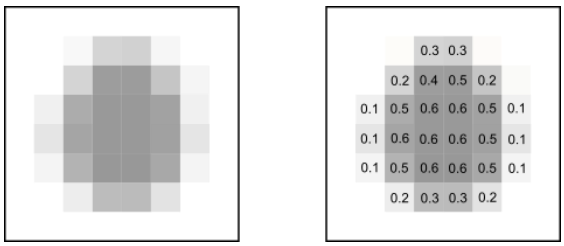

Figura 6.3. Una imatge és una matriu de píxels amb valors de 0 a 255 normalitzats entre 0 i 1

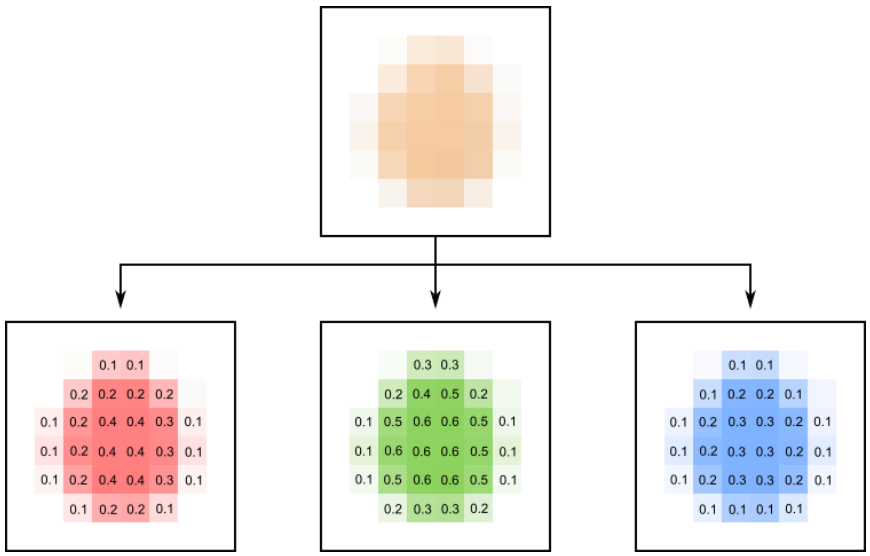

Figura 6.4. Si la imatge és RGB, estarà formada pels tres colors bàsics: vermell, verd i blau

Tot i que en el projecte s'han utilitzat imatges en color, per simplificar les explicacions se suposaran que les imatges són en blanc i negre.

6.2.2 Capes ocultes

Si ens fixem en la Figura 6.2, veiem que la xarxa neuronal està formada per blocs idèntics, cadascun d'ells contenint 4 capes convolucionals i un *max-pooling* al final. Anem a intentar explicar què són aquestes dues operacions:

Operació de convolució

La principal diferència entre una capa densament connectada (xarxes neuronals tradicionals) i una capa especialitzada en la operació de convolució, que anomenarem capa convolucional, és que la capa densa aprèn patrons globals en el seu espai global d'entrada, mentre que la capa convolucional aprèn patrons locals dins de la imatge.

De manera intuïtiva podríem dir que gràcies a l'operació de convolució, les CNN poden detectar característiques visuals en les imatges, com ara línies, formes, textures, etc. Aquesta és una propietat molt important perquè un cop apresa una característica en un punt concret de la imatge, la pot reconèixer després en qualsevol punt d'aquesta mateixa. En canvi, una xarxa neuronal densament connectada ha d'aprendre el patró novament si aquesta apareix en una nova localització de la imatge.

L'operació de convolució consisteix a prendre "grups de píxels propers" de la imatge d'entrada i anar realitzant el producte escalar amb una petita matriu que s'anomena filtre (en anglès *kernel*). Aquest filtre recorre totes les neurones d'entrada (d'esquerra a dreta i de dalt a baix) i genera una nova matriu de sortida, que en definitiva serà la nostra nova capa de neurones. Per cada posició del filtre hi ha una neurona en la capa següent que processa aquesta informació.

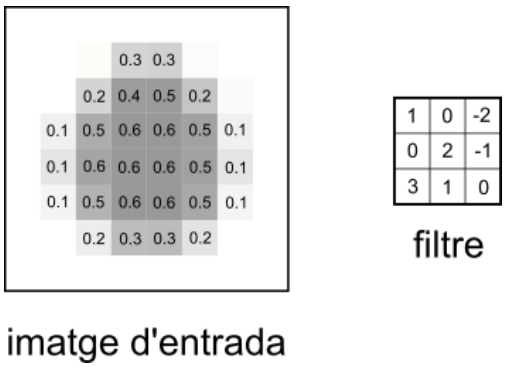

Figura 6.5. En aquesta figura, la imatge d'entrada té una mida de 8x8 píxels. A la xarxa del projecte les imatges d'entrada són de 128x128 px

Aquest filtre inicialment prendrà valors aleatoris i a mesura que es vagi entrenant s'aniran ajustant. També cal destacar que, en el cas de les imatges en color (tal com passa en el projecte), el filtre realment seria de 3x3x3. És a dir, un filtre amb 3 filtres (un per

cada color) de mida 3x3. Després aquests 3 filtres se sumen i conformen una sortida (com si fos un sol canal).

Un filtre només és capaç de detectar una característica concreta en una imatge. Per tant, per poder realitzar un reconeixement d'imatges es proposa utilitzar diversos filtres al mateix temps, un per cada característica que vulguem detectar. És per això que en la nostra xarxa neuronal, en cadascun dels 4 blocs es posen 3 capes convolucionals de 32 filtres cadascuna (normalment se sol prendre grups de 32, 64, 128, etc.)., amb la qual cosa realment obtenim 32 matrius de sortida (aquest conjunt es coneix com *feature map*).

A continuació podem observar com funciona: el filtre realitza el producte matricial amb la imatge d'entrada i desplaçant-se a 1 píxel d'esquerra a dreta i de dalt a baix, i va generant una nova matriu que compon el mapa de característiques (*feature map*). El tret particular i més important de les xarxes convolucionals és que s'utilitza el mateix filtre (i els mateixos pesos) per totes les neurones de la capa següent. Això permet reduir dràsticament el nombre de paràmetres que tindria la xarxa neuronal si no ho féssim així.

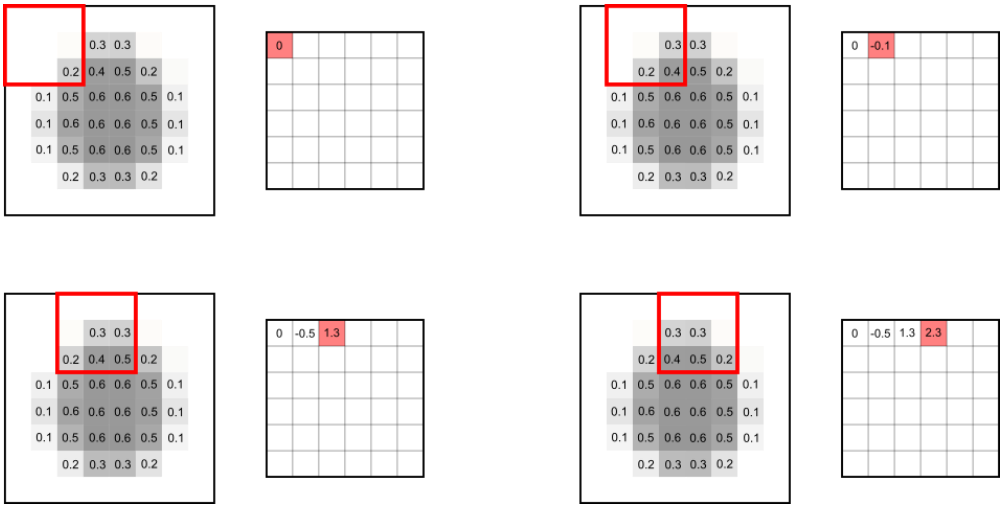

Figura 6.6. Operació de convolució

A mesura que anem desplaçant el filtre anem obtenint una nova imatge filtrada. En aquesta primera convolució i seguint amb l'exemple anterior, és com si obtinguéssim 32 imatges filtrades noves. Aquestes imatges noves el que estan dibuixant són certes característiques de la imatge original. Això ajudarà en el futur a poder distingir un objecte de l'altre.

La imatge realitza una convolució amb un filtre i aplica una funció d'activació, en el nostre cas ReLu (*Rectifier Linear Unit*). És la funció d'activació més utilitzada per aquest tipus de xarxes i consisteix en f(x) = max(0,x), de manera que s'eliminen els valors negatius.

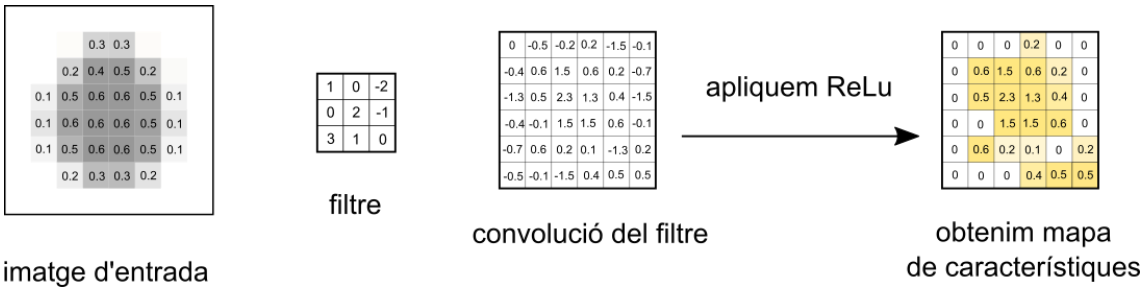

Figura 6.7. La imatge realitza una convolució amb el filtre i finalment aplica una funció d'activació ReLu

Operació de pooling (agrupament)

A més de les capes convolucionals que s'han explicat, les xarxes neuronals convolucionals disposen també de capes de *pooling*, que es solen aplicar després de les capes convolucionals. Aquestes serveixen per fer una simplificació de la informació recollida per la capa convolucional i crear així una versió resumida, mantenint les característiques més importants. Si no féssim cap *pooling* i només féssim convolucions, el nombre de neurones de la xarxa creixeria enormement (i això implica un major processament).

Existeixen diverses maneres de condensar la informació, però la més habitual és el *max-pooling*. En aquest projecte, després d'un seguit de convolucions, s'ha fet un *max-pooling* de mida 2x2.

Això significa que recorrem cada una de les nostres 32 imatges obtingudes anteriorment de 128x128 px d'esquerra a dreta i de dalt a baix, però en lloc d'agafar els píxels d'un en un, agafarem una zona de 2x2 (2 d'alçada per 2 d'amplada = 4 px) i anirem preservant el valor més elevat d'entre aquests 4 píxels (d'aquí el nom de *max-pooling*). En aquest cas, utilitzant una finestra de 2x2, la imatge resultant és reduïda a la meitat i quedarà de 64x64 px.

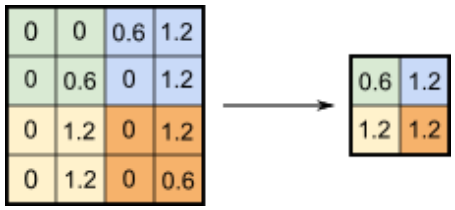

Figura 6.8. Agafem regions de 4 px i ens quedem amb el valor més elevat. Referència [6]

Després d'aquest procés de *max-pooling* ens quedaran 32 imatges de 64x64 que segueixen emmagatzemant la informació més important per detectar les característiques desitjades.

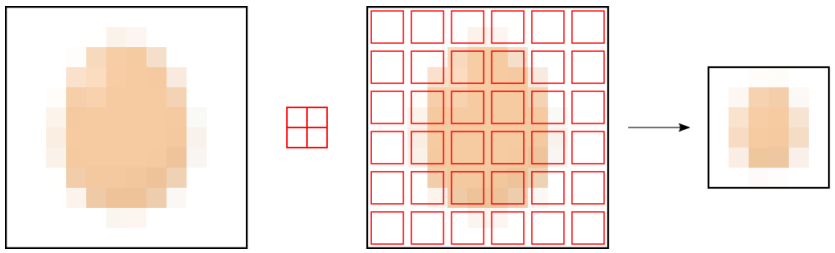

Figura 6.9. Després del *max-pooling* visualment veiem que hi ha la mateixa informació, però condensada, ja que l'hem reduït a la meitat.

▪ Bloc de convolucions + pooling

La primera convolució és capaç de detectar característiques primitives com línies, corbes o textures.

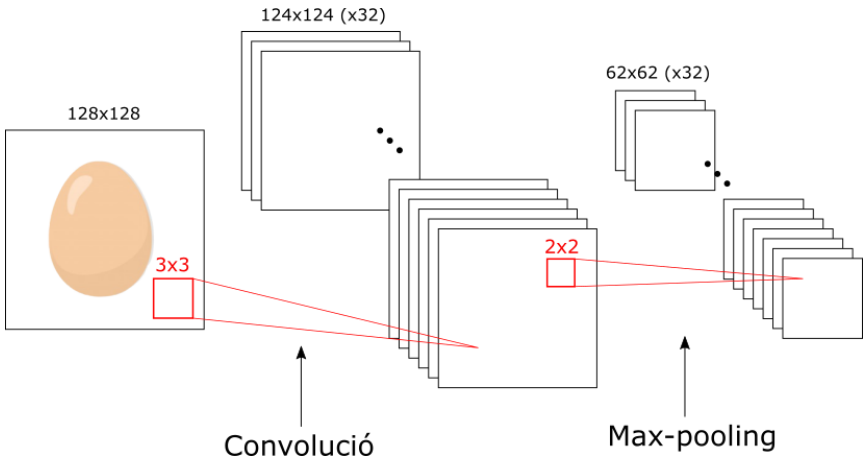

Figura 6.10. Primera capa convolucional i *max-pooling*

A mesura que afegim més capes convolucionals, els mapes de característiques seran capaços de reconèixer formes més complexes, i el conjunt total de capes convolucionals podrà arribar a "veure".

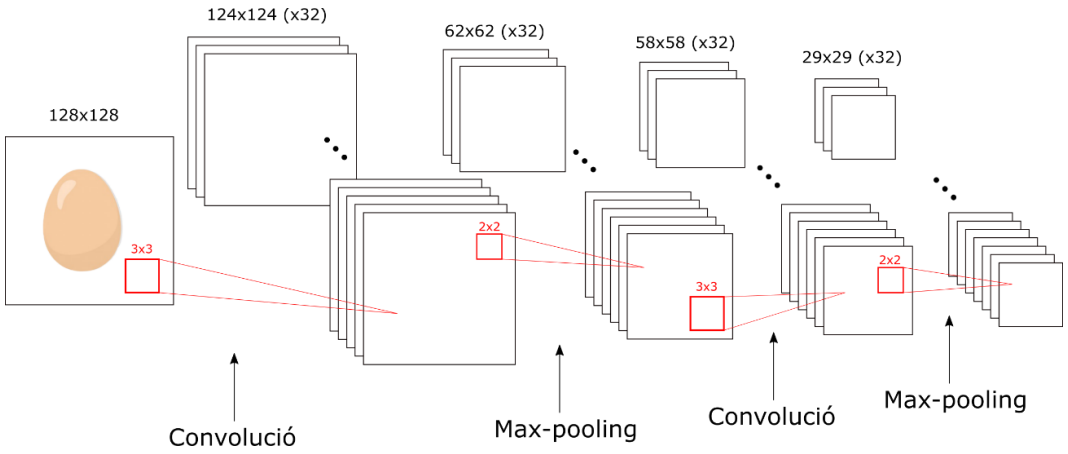

Figura 6.11. Les xarxes convolucionals es composen de vàries convolucions, fent així la xarxa cada cop més profunda

6.2.3 Sortides i connexió amb una xarxa neuronal densa

Finalment, per prendre una decisió (en el cas de classificació) o per retornar un valor (en el cas de regressió) ens cal connectar l'última capa de la xarxa neuronal convolucional amb una xarxa neuronal densament connectada, per tal que ens tradueixi la informació.

Per fer aquesta connexió el que es fa és agafar l'última capa de sortida i aplanar-la (*flatten layer*), transformant-la així d'un tensor 3D a un vector 1D. Seguidament, en funció de la tasca que volem fer hi podem connectar una o més capes de neurones.

Predicció de la bounding box

Després d'aplanar la última capa de la branca principal, es connecta la sortida a una capa densament connectada amb 256 neurones i, a continuació, a aquesta se li connecta una capa altra capa de 4 neurones. Així doncs, s'obté com a resultat un vector amb les 4 coordenades de la *bounding box* que indica la localització de l'ou:

$$\boldsymbol{b} = (x, y, w, h) \tag{6.1}$$

Predicció del pes

De l'última capa de la branca principal hi ha una segona branca encarregada de predir el pes. De la mateixa manera que en l'anterior, es connecta aquesta última capa aplanada amb una altra capa densament connectada de 256 neurones, després una de 4 neurones i finalment una sola neurona, que indicarà el pes de l'ou:

$$\boldsymbol{p} \tag{6.2}$$

6.3 Entrenament

Una xarxa neuronal no és res més que un model que intenta relacionar una entrada (imatge d'un ou en aquest cas) amb una o diverses sortides (coordenades de la bounding box i pes de l'ou). Per poder fer aquesta relació, el model ha d'aprendre els seus paràmetres (pesos) observant moltes mostres i les seves corresponents solucions durant el procés d'entrenament. El procés d'entrenament, per tant, consisteix a trobar els valors adequats d'aquests pesos.

Però, com s'aconsegueix l'aprenentatge d'aquests paràmetres? Es tracta d'uns paràmetres "entrenables", que s'inicialitzen per defecte amb valors aleatoris. A partir d'aquí, el seu valor es va ajustant gradualment. El resultat és que aquests paràmetres contenen informació apresa per la xarxa pel fet d'haver estat exposada a les dades d'entrenament. Per tant, en aquest context, aprendre significa trobar un conjunt de valors pels quals els paràmetres de totes les capes d'una xarxa, de manera que la xarxa "mapegi" correctament mostres d'entrada a les seves etiquetes associades.

Ara bé, en el cas de les CNN hem d'ajustar el valor dels pesos dels diferents filtres. Això és un gran avantatge al moment de l'aprenentatge, ja que com hem vist cada filtre es d'una mida reduïda, en el nostre exemple en la primera convolució és de mida 3x3, això són tan sols 9 paràmetres que hem d'ajustar en 32 filtres, donant un total de 288 paràmetres.

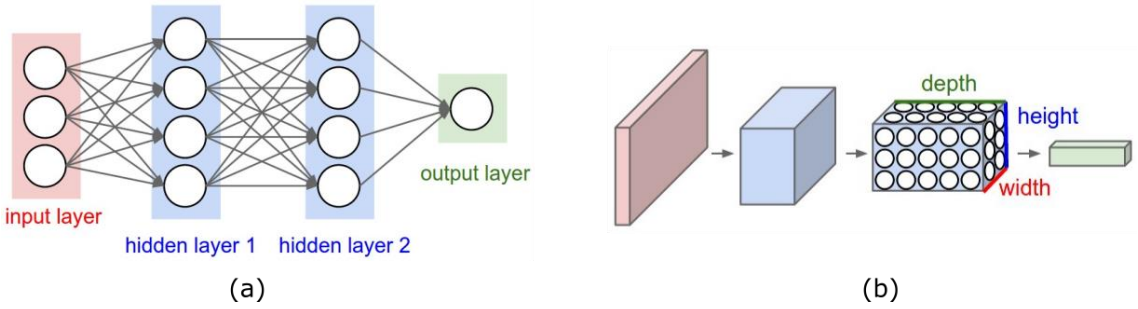

Figura 6.12. (a) Xarxa neuronal densament connectada (o tradicional) respecte (b) xarxa neuronal convolucional (CNN)

Per controlar la sortida d'una xarxa neuronal, cal que es pugui mesurar quant de lluny està la sortida del que s'esperava. Aquest és el treball de la funció de pèrdua de la xarxa (*function loss* en anglès). La funció de pèrdua agafa les prediccions de la xarxa i el valor verdader (el que esperaríem que la xarxa produís) de la etiqueta i calcula un error comès en una mostra d'entrada específica

En aquest valor està el quid de la qüestió. Es tracta d'aprofitar aquesta mesura de l'error comès com a senyal de retroalimentació del sistema per ajustar el valor dels paràmetres, en una direcció que disminueixi el càlcul d'error per la mostra actual (en realitat, per totes les mostres futures).

Aquest ajust és precisament el treball que realitza l'optimitzador, que implementa aquesta "retropropagació" d'aquest error per ajustar els paràmetres de la forma que hem dit. Aquest algoritme és l'element clau de l'aprenentatge i rep el nom de *Backpropagation*.

Com hem comentat anteriorment, en general, inicialment els pesos de la xara se'ls assignen valors aleatoris, amb la qual cosa l'error calculat al principi és alt perquè la xarxa comença fent estimacions aleatòries. Però mica en mica, amb cada mostra d'entrada que processa la xarxa, els pesos s'ajusten una mica en la direcció correcta i l'error calculat va disminuint progressivament. Aquest és el cicle d'entrenament que, repetit un número suficient de vegades, produeix valors de pes que minimitzen el resultat de la funció de pèrdua.

6.3.1 Configuració de l'entrenament

Tant per la predicció de la *bounding box* com per la predicció del pes s'utilitza la mateixa funció de pèrdua: MSE (de l'anglès *mean squared error*). Tal i com el seu nom indica, la

seva pèrdua es calcula agafant la mitjana de les diferències al quadrat entre els valors reals i els pronosticats.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2 \quad (6.3)$$

L'algoritme utilitzat per actualitzar els pesos és l'optimitzador Adam [9] amb un rati d'aprenentatge (en anglès *learning rate*) de $1 \cdot 10^{-4}$.

Pel que fa al nombre d'iteracions, com la majoria de paràmetres, no hi ha una única solució correcta ni una manera concreta de saber quin és el valor adequat. Simplement és provant i canviant els paràmetres i veient si milloren els resultats que es troben els valors correctes. Així doncs, després d'uns quants entrenaments s'ha decidit fer 50 iteracions, amb una mida dels lots de 32.

Tota la implementació de la xarxa neuronal s'ha fet amb la llibreria Tensorflow [10].

6.4 Data Augmentation

Normalment les xarxes neuronals tenen una gran quantitat de paràmetres a ajustar per poder generalitzar o aprendre característiques d'un *dataset*. Un dels components bàsics per aconseguir això són els enormes *datasets* amb imatges que alimenten l'algoritme; en el nostre cas, però, aquestes imatges són limitades, ja que s'han hagut de capturar i etiquetar manualment.

L'augmentació de dades (*data augmentation*) és una tècnica que ens permet augmentar el nombre d'imatges del nostre *dataset* per millorar la precisió, la generalització i controlar l'*overfitting* (sobreentrenament).

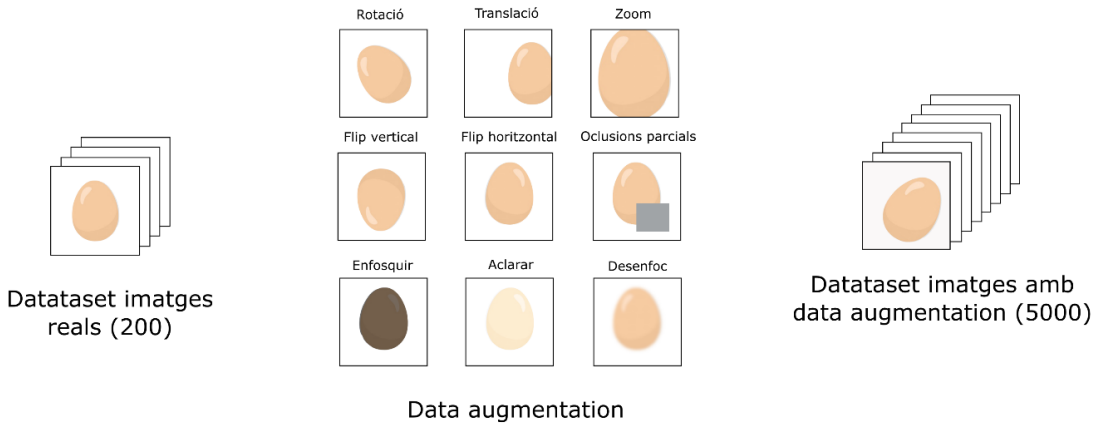

Figura 6.13. Partint d'un dataset amb només 200 imatges d'ous reals, es pot arribar a crear un dataset amb 5000 imatges afegint petites variacions.

Així doncs, l'augmentació de dades és la generació artificial de dades per mitjà de pertorbacions i petits canvis en les dades originals. Això ens permet augmentar tan en mida com en diversitat el nostre conjunt de dades.

Les transformacions que s'han realitzat són les següents:

- Rotacions
- Translacions
- Reflex horitzontal i vertical
- Oclusions parcials
- Variacions en la lluminositat i contrast
- Desenfocaments

Hi ha dues maneres de fer augmentació de dades: o bé agafar totes les imatges, fer les transformacions i desar-les en una carpeta (per després passar-les a través de la xarxa), o bé fer *data augmentation* al moment; en aquest segon cas no s'arriba a emmagatzemar la imatge, sinó que es fa la transformació i directament després es passa per la xarxa (sense desar-la). Això ajuda a ocupar menys memòria, però per altra banda alenteix el procés d'entrenament. Tot i que es van provar les dues maneres, finalment es va decidir utilitzar la primera.

A continuació es mostren alguns exemples d'imatges augmentades:

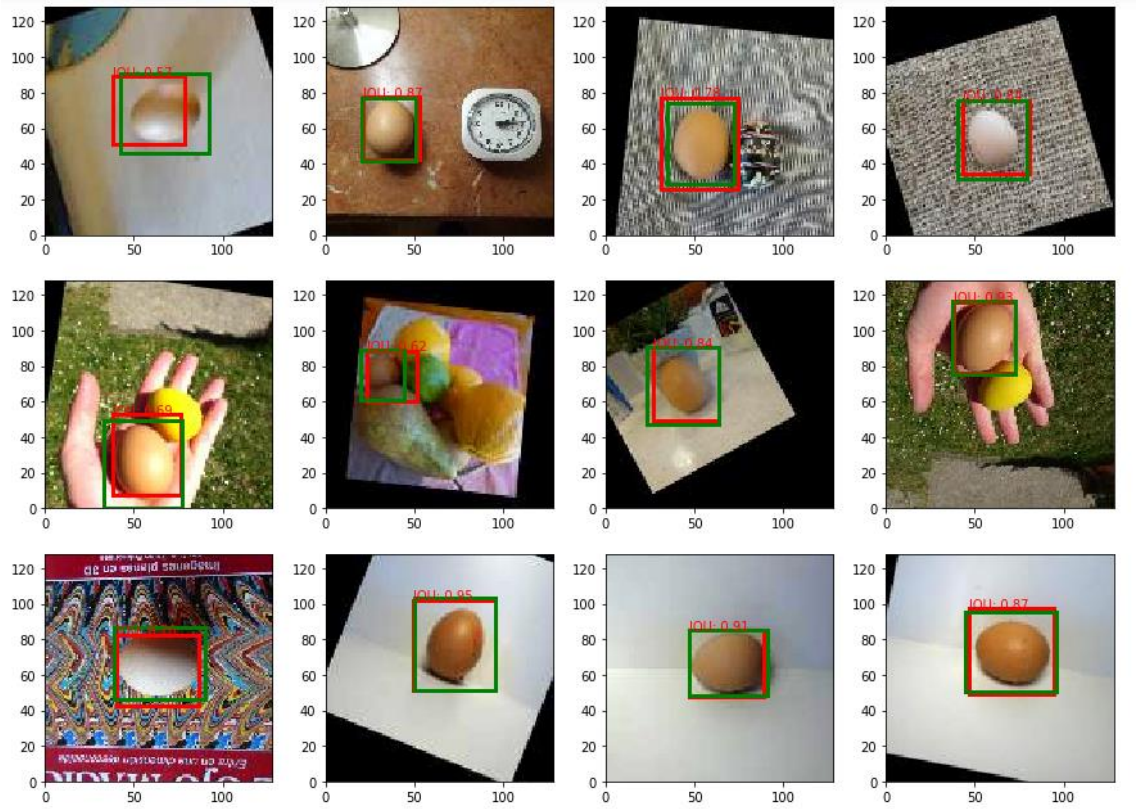

Figura 6.14. Mostra d'algunes imatges amb petites transformacions

Per dur a terme aquestes petites variacions en les imatges s'ha utilitzat la llibreria imgaug [7]. Aquesta permet fer un gran nombre de transformacions, combinar-les en una mateixa imatge i executar-les de manera aleatòria molt fàcilment.

7. Interfície gràfica

En aquest apartat s'explicarà el procés seguit per la creació de la interfície gràfica. En la primera secció es farà una breu anàlisi del què es busca aconseguir i de què és del que ha de disposar aquesta interfície. En la segona secció es parlarà del disseny que s'ha proposat i finalment, en la darrera secció, es comentaran els aspectes d'implementació.

7.1 Anàlisi

Com ja s'ha comentat als objectius del treball, a més de crear el model predictiu de les imatges dels ous, també es busca adaptar el sistema de manera que qualsevol usuari pugui fer prediccions d'una manera fàcil i ràpida.

La idea és poder carregar imatges al servidor, que aquest executi la xarxa neuronal i faci una predicció de la imatge carregada, i retorni el resultat a l'usuari amb la localització de l'ou i el pes predit.

Ara bé, per aquest treball i per fer-nos una idea de com funciona la xarxa en una aplicació real ja n'hi haurà suficient.

7.2 Disseny

Cal pensar en una estructura que permeti unir la xarxa neuronal entrenada amb una interfície on l'usuari pugui carregar imatges i rebre'n uns resultats d'aquestes. Per tal de resoldre aquest problema es decideix fer-ho mitjançant una *Web App*.

Ha de ser una interfície molt simple, pràctica i fàcil d'utilitzar, ja que recordem que volem agilitzar el procés de pesatge dels ous. També hem de pensar en el tipus d'usuari que l'utilitzarà (operaris sense grans coneixements ni d'intel·ligència artificial ni de programació, per tant ha de ser intuïtiva).

No caldrà mostrar càlculs realitzats ni estadístiques (o si més no haurà de ser opcional) ja que no és el que es busca: la finalitat és que simplement torni la localització i el pes, i ho faci el més ràpid i precís possible.

Així doncs, s'escull utilitzar una *Web App* per fer el prototip de funcionament. Una *Web App* és una aplicació o programa que es carrega en un servidor i s'executa en el navegador (no requereix de la instal·lació de cap software).

7.2.1 Esquema

En tota *Web App* hi ha dos elements bàsics: el servidor i el navegador. Més enllà d'aquests, però, hi ha una estructura que els engloba i que es pot resumir de la següent manera:

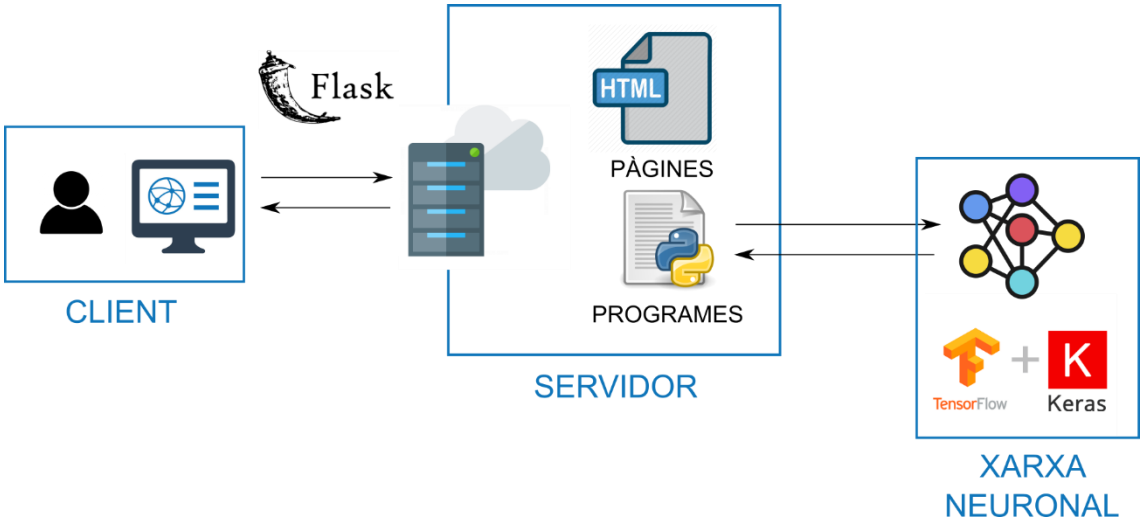

Figura 7.1. Esquema de l'estructura d'una *Web App*

Elements principals:

▪ **Costat client**

- **Usuari:** és la persona que interactua amb la *Web App*. En el nostre cas serà l'operari que estarà controlant els ous de gallina abans de classificar-los, i haurà de fer fotos d'aquests i llavors carregar-les al servidor a través del navegador.
- **Navegador:** és on s'executa la *Web App* i actuarà com a intermediari entre el servidor i l'usuari. Aquest serà l'encarregat d'enviar les imatges al servidor (petició) i alhora rebre els resultats (localització i pes de l'ou) i mostrar-los en pantalla.

▪ **Costat servidor**

Un servidor es defineix com un dispositiu o programa dedicat a gestionar recursos web. En el cas de les *Web App*, executa les peticions rebudes pel client. Aquesta petició és retornada en forma de codi web executable en el navegador.

- **Pàgines:** són la interfície en sí (aspecte) que mostra el navegador, és a dir, el disseny des de la distribució, passant pel tipus de lletra fins als colors. Aquestes estan escrites, principalment, en llenguatge HTML, CSS i JavaScript. Una *Web App* pot tenir vàries pàgines les quals mostren en el navegador.
- **Funcions:** són les encarregades de processar la informació rebuda del client, i donar una resposta. La resposta s'envia dins del codi de la pàgina perquè pugui ser mostrada.

7.2.2 Pantalles

Pantalla principal

Ha de permetre carregar la imatge i mostrar una vista prèvia (que realment hàgim seleccionat la imatge que volíem).

En aquesta pantalla també pot sortir informació que sobre com ha de ser la imatge per tal que la predicció sigui la millor possible (distància a la qual s'ha de tirar la foto, format de la imatge, mida màxima, etc.).

Figura 7.2. Pantalla principal

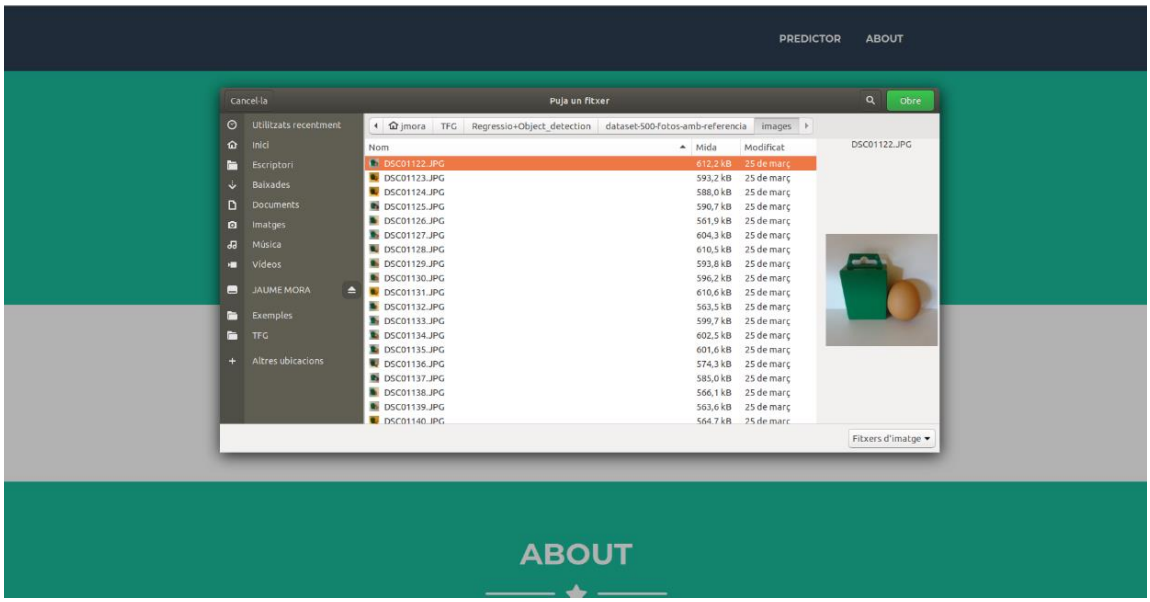

Figura 7.3. Selecció d'una imatge

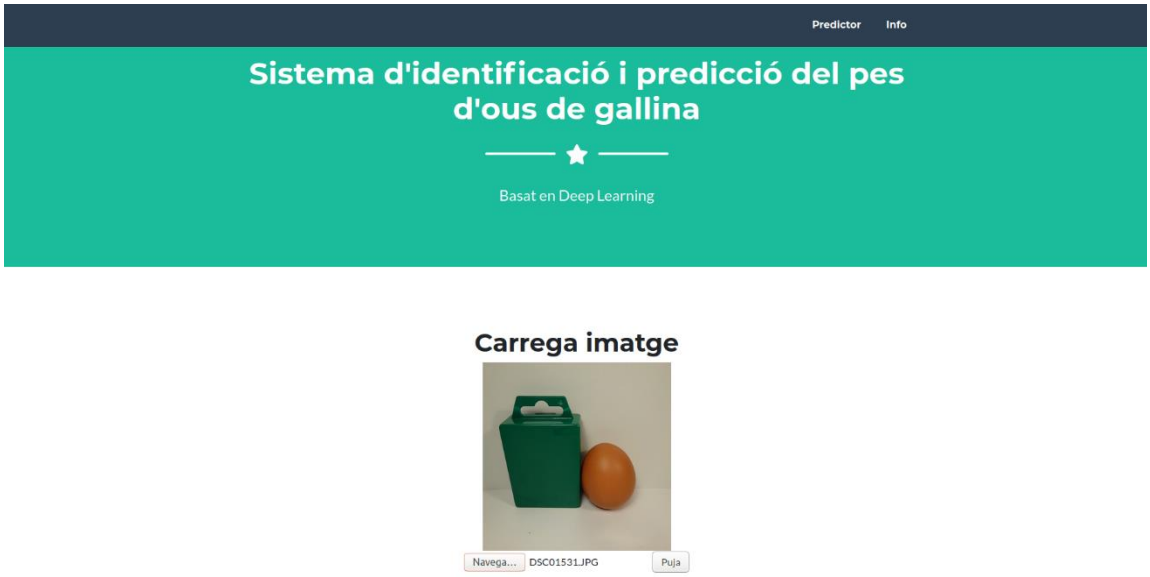

Figura 7.4. Vista prèvia de la imatge carregada

Dins aquesta mateixa pantalla principal, a la part de baix hi ha una breu explicació del projecte i la seva finalitat. Aquesta pantalla en producció segurament no hi hauria de ser, però al ser un prototipus s'ha deixat com a informació extra.

Figura 7.5. Secció de la pantalla principal amb l'explicació del projecte

Pantalla resultats

En aquesta pantalla s'ha de mostrar la imatge que hem carregat, emmarcada amb la *bounding box* predita i a sota el pes de l'ou que apareix en la imatge. Com s'ha explicat anteriorment, es vol que la informació sigui clara i concisa.

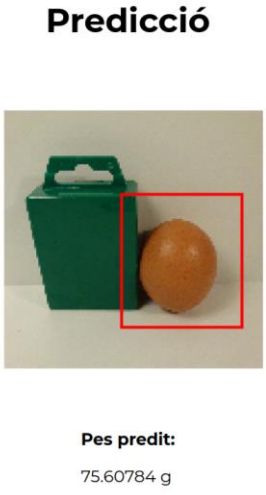

Figura 7.6. Resultat de la predicció

Pantalla error

En cas que hi hagi hagut algun error de comunicació entre el client i el servidor que impedeixi mostrar el resultat correctament, cal una pantalla per explicar-ho.

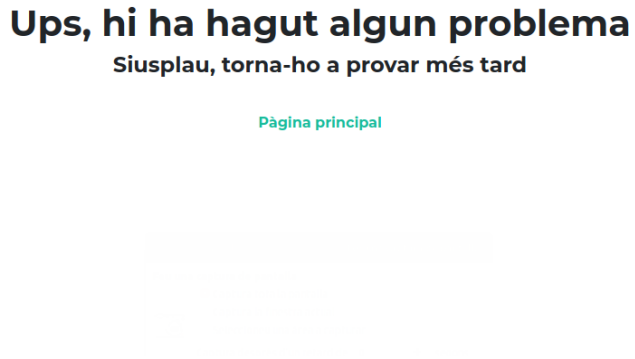

Figura 7.7. Pantalla error

7.3 Implementació

7.3.1 Framework

Per crear l'aplicació web es va escollir Flask [8] com a entorn de treball (*framework*) ja que utilitza Python (llenguatge utilitzat en tot el projecte), és de codi obert i a més permet crear pàgines web d'una manera fàcil i ràpida.

7.3.2 Estructura de carpetes

Tot projecte amb Flask acostuma a presentar una estructura semblant. En aquest cas s'ha organitzat de la següent manera:

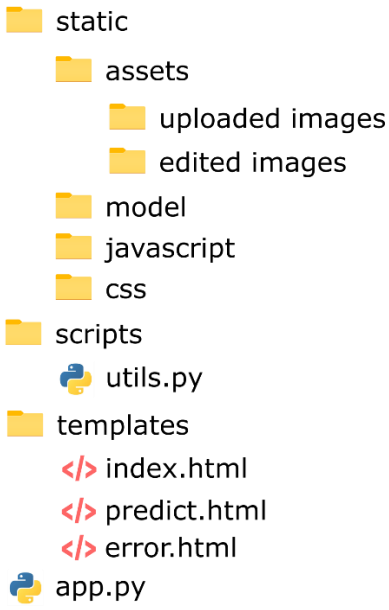

Figura 7.8. Estructura de carpetes de l'aplicació web

App.py

Aquest fitxer s'encarregarà de crear l'objecte aplicació i gestionarà les peticions del client, mostrant les pantalles (fitxer HTML) que toca en cada cas.

Static

En aquesta carpeta hi haurà els fitxers estàtics. Aquesta carpeta s'ha subdividit en tres carpetes:

- Assets: conté dues carpetes, *uploaded images* (carpeta amb les imatges que carrega l'usuari) i *edited images* (carpeta amb les imatges amb la *bounding box* dibuixada). Les *edited images* es generen a partir de les *uploaded images* un cop feta la predicció.
- Model: aquesta carpeta conté el model neuronal entrenat.
- Css: conté els estils de la pàgina (colors, fonts de text, espaiats, etc.)
- Javascript: conté els displays dinàmics (efectes i transicions)

Templates

En aquesta carpeta hi haurà les quatre pàgines HTML explicades en l'apartat anterior.

Scripts

Aquí s'hi troben els scripts en Python, amb les funcions a dins que ens permetran carregar les imatges, processar-les, avisar al model neuronal perquè faci una predicció i obtenir els resultats.

7.3.3 Funcionament

Els passos que seguirà l'aplicació des de que l'usuari carrega una imatge, fins que rep el resultat seran els següents:

1. **Selecció imatge i previsualització**

   Quan l'usuari clica el botó "Tria imatge" de la pantalla principal, s'obre l'explorador de fitxers del sistema per tal que l'usuari navegui fins a la carpeta on es troba la imatge que vol carregar.

   Un cop seleccionada, clica "D'acord" i automàticament es mostra una previsualització de la imatge carregada (per tal que l'usuari pugui veure que realment ha seleccionat la imatge que volia).

2. **Comprovacions i càrrega de la imatge**

   Després que l'usuari cliqui "Carrega imatge" es comproven un seguit d'elements: que no hi hagi cap imatge seleccionada, el format de la imatge, la mida, etc. Si tot això és correcte, es guarda aquella imatge en la carpeta *uploaded images*.

3. **Processament imatge**

   A continuació, s'avisa la funció *process image* que agafa la imatge guardada en la carpeta i la "prepara" pel model neuronal.

   Això bàsicament consisteix en fer la imatge més petita i transformar-la en un *array* de píxels de manera que el model neuronal pugui operar amb ella.

4. **Execució del model i obtenció dels resultats**

   S'executa el model (amb els pesos adquirits durant l'entrenament) i s'obtenen els resultats de la *bounding box* (vector de quatre coordenades) i el pes. Seguidament es crida a la funció "plot_bbox" i el que fa és dibuixar la bounding box obtinguda sobre la imatge (mitjançant la llibreria matplotlib [13] de Python) i guardar-la en la carpeta *edited images*. Després s'envia la informació amb la localització de la imatge editada i el pes obtingut al "render" en forma de missatge.

5. **Mostra de la predicció**

Finalment es mostra la imatge amb la *bounding box* pintada i el pes predit.

En cas que hi hagi algun error en algun d'aquests passos (el format de la imatge no és el que toca, la imatge pesa massa i l'aplicació triga molt en guardar-la, el model té algun error en el procés de predicció, etc.) es mostrarà automàticament la pantalla d'error.

8. Resultats

En aquest apartat s'analitzaran els resultats obtinguts amb el model neuronal entrenat. En la primera secció s'explicarà quines mètriques de mesura s'han fet servir i com es calcula l'error. En la segona secció es mostraran els resultats obtinguts pel model en funció del *dataset* amb què és entrenat. Finalment, en la última secció, es compararan els resultats obtinguts i s'intentarà entendre el perquè d'aquests.

8.1 Mètriques per avaluar el model

Per cada sortida de la xarxa s'utilitza una mètrica diferent per mesurar l'error. A continuació es fa una breu explicació sobre cadascuna d'elles.

Intersection Over Union (IoU)

Per avaluar l'error comès en la predicció de la localització de l'ou, s'utilitzarà la mètrica *Intersection Over Union* (intersecció sobre la unió, IoU), mètrica utilitzada en les tasques de detecció d'objectes. L'IoU treballa amb les *bounding boxes* reals (etiquetades) i les predites pel model, i a partir d'aquí calcula una mesura de la precisió.

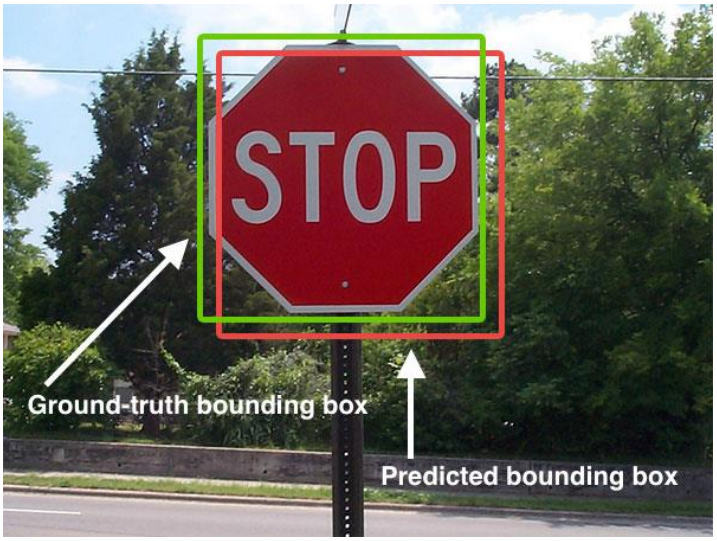

Figura 8.1. La *bounding box* predita està dibuixada en vermell mentre la real està en verd. L'objectiu és calcular la IoU entre aquestes dues *bounding boxes*.

Així doncs, la *Intersection Over Union* vindrà determinada per la següent equació:

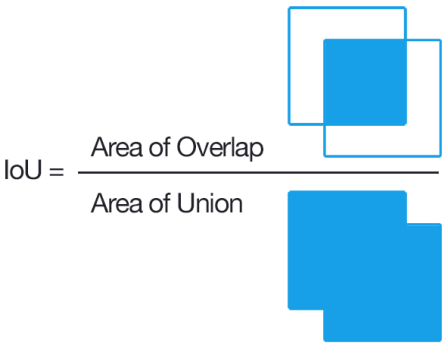

Figura 8.2. Per calcular la IoU s'ha de dividir l'àrea sobreposada entre les dues *bounding boxes* per l'àrea d'unió. Imatge extreta de [9]

Una $IoU > 0.5$ és considerada una bona predicció. En el cas que les dues *bounding boxes* fossin exactament iguals, llavors l'IoU seria 1.

Quan fem problemes de classificació o de regressió és molt fàcil mesurar quan una cosa està ben predita o no (els valors han de ser iguals), però en problemes de detecció d'objectes no és tan fàcil. En realitat, les coordenades de la *bounding box* predita no coincidiran mai exactament amb les de la *bounding box* real, però això no significa que la predicció sigui dolenta. Així doncs, aquesta mètrica mesura la superposició, i encara que les coordenades no siguin exactament iguals, si una gran part de les dues *bounding boxes* coincideix serà una bona predicció.

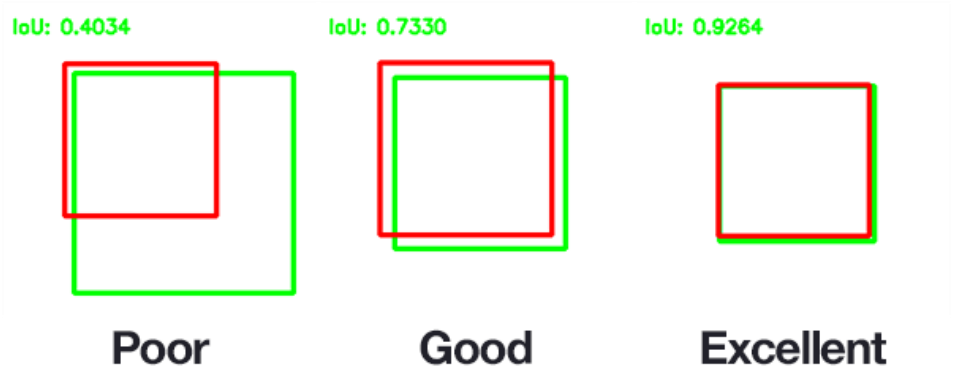

Figura 8.3. Exemple de càlcul de IoU per vàries *bounding boxes*, extreta de [9]

Tal com es pot veure en la imatge anterior, com més es superposen les *bounding boxes*, major és la IoU (no es té en compte si la superposició és interna o externa).

Mean Squared Error

En el cas de la predicció del pes de l'ou, s'utilitzarà com a mètrica d'avaluació la mateixa funció que es fa servir com a *loss function* per entrenar el model, vista en l'apartat 6.3.1.

Per tal de visualitzar els resultats, es farà un histograma amb els pesos reals dels ous del *dataset* i amb els pesos predits per la xarxa neuronal, així com també s'analitzarà la mitjana i la desviació típica de cada distribució.

8.2 Entrenament i prova dels models

Per tal de comparar resultats, s'ha creat dos models neuronals amb la mateixa estructura de capes i amb els mateixos paràmetres d'entrenament, però en un cas s'ha entrenat amb imatges d'ous amb referència (una caixa verda al costat de l'ou que permet saber la mida relativa) mentre que en l'altre cas s'ha entrenat amb imatges d'ous sols.

El que es farà primer serà analitzar l'entrenament de cadascun dels models, és a dir, si les gràfiques de *training* i *validation* tenen la forma esperada i es pot observar que les xarxes estan aprenent. Després s'utilitzarà cada model per fer prediccions tant en imatges d'ous amb referència, com imatges amb ous sense referència, i es compararan els resultats obtinguts amb cada model. El que funcioni millor serà el que s'utilitzarà per l'aplicació web.

8.2.1 Model 1: entrenat amb referència

Aquest model ha estat entrenat amb 5000 imatges *augmented*. Ara bé, aquestes 5000 imatges han estat generades a partir de 250 fotos d'ous amb referència i aquestes, al mateix temps, només contenen 50 ous diferents (5 fotos per ou).

A continuació, aquest *dataset* de 5000 imatges s'ha separat amb imatges d'entrenament i de validació com ja s'ha explicat a l'apartat 5.3, les dades d'entrenament serveix perquè el model aprengui, mentre que les de validació serveixen per ajustar-lo. Les de test no les ha vist mai el model.

D'aquesta manera, per un entrenament amb 50 iteracions (*epochs*) les gràfiques per cadascuna de les sortides són les següents:

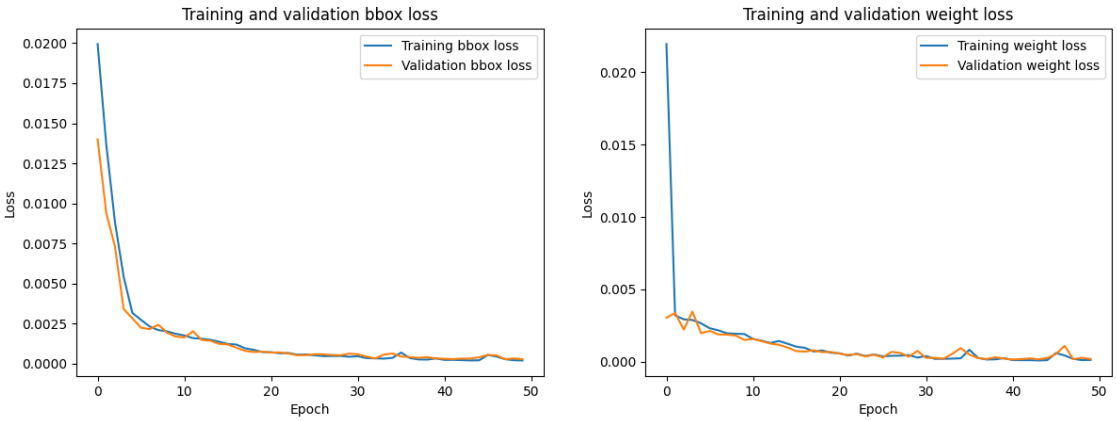

Figura 8.4. Gràfica de l'error de *training* i *validation* tant per la sortida de la *bounding box* (esquerra) com pel pes (dreta)

En les dues gràfiques s'observa que l'error va disminuint, per tant podem considerar que el model està aprenent les característiques d'un ou.

A continuació es testejarà el model amb dos *datasets* diferents, un amb imatges d'ous amb referència i un altre amb imatges amb ous sense referència. Aquesta fase de test consisteix en el fet que el model fa prediccions de fotos que no ha vist mai i s'avalua l'error comès.

Test amb *dataset* d'ous amb referència

Aquest *dataset* està format per 50 fotos d'ous amb referència, cap de les quals ha servit ni per generar *augmented images* ni per avaluar el model. A continuació es mostren els resultats de les prediccions:

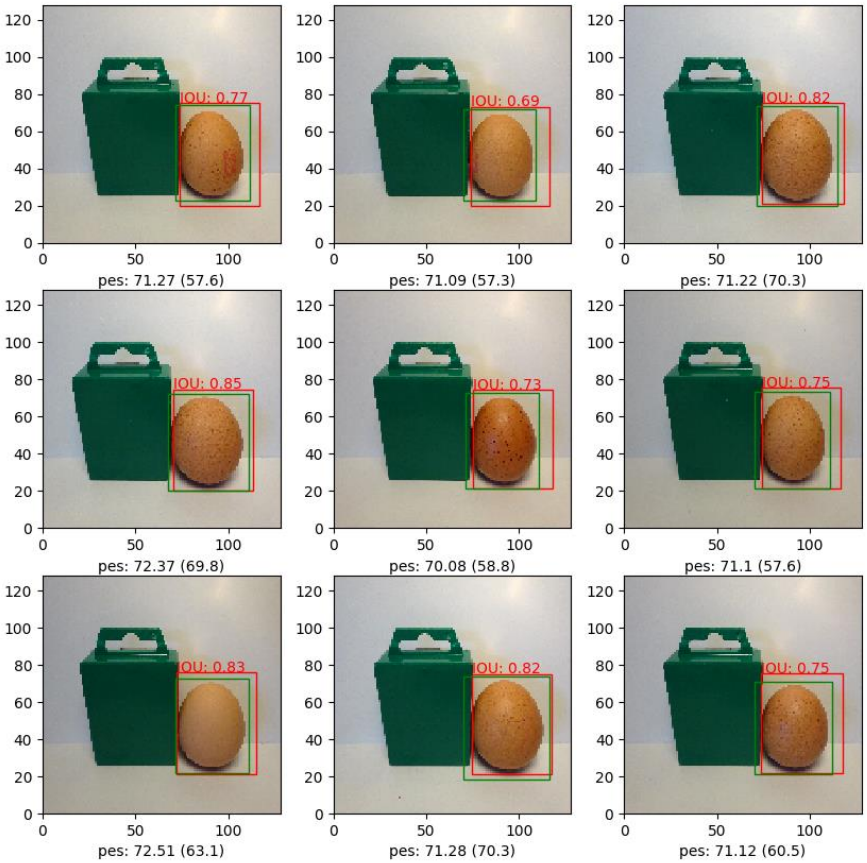

Figura 8.5. Mostra d'algunes prediccions. La *bounding box* real (en verd), la predita amb la mesura de l'IoU (en vermell) i a sota de cada imatge el pes predit i el pes real (aquest últim entre parèntesis).

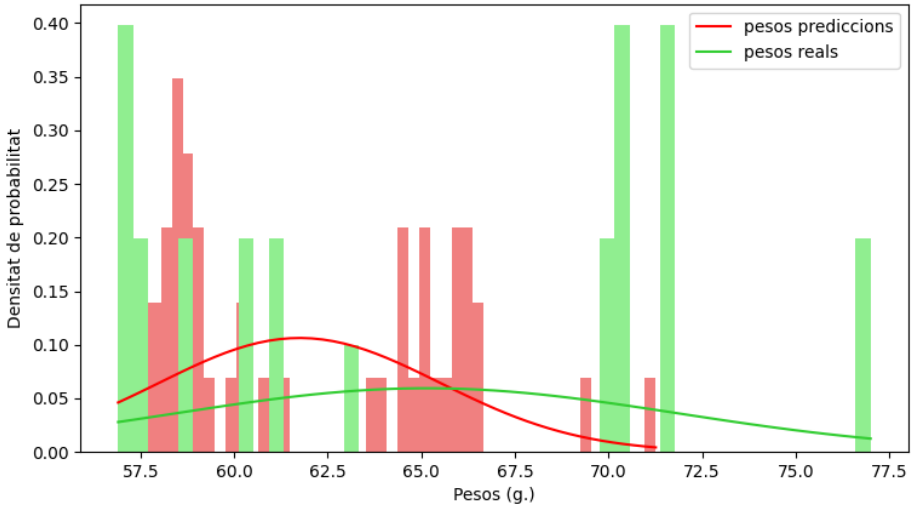

Figura 8.6. Distribucions dels pesos reals (en verd) en comparació amb els predits (en vermell).

	Distribució real	Distribució predita
Mitjana del pes(g)	65,15	61,75
Desviació típica (σ)	6,71	3,75

Taula 8.1. Mitjanes i desviacions típiques de les distribucions

IoU mitjà de la predicció = 0,81

Error absolut mitjà[6] del pes = 3,86 g

Test amb *dataset* d'ous sense referència

Aquest *dataset* està format també per 50 fotos d'ous amb referència, cap de les quals ha servit ni per generar *augmented images*. A continuació els resultats de les prediccions:

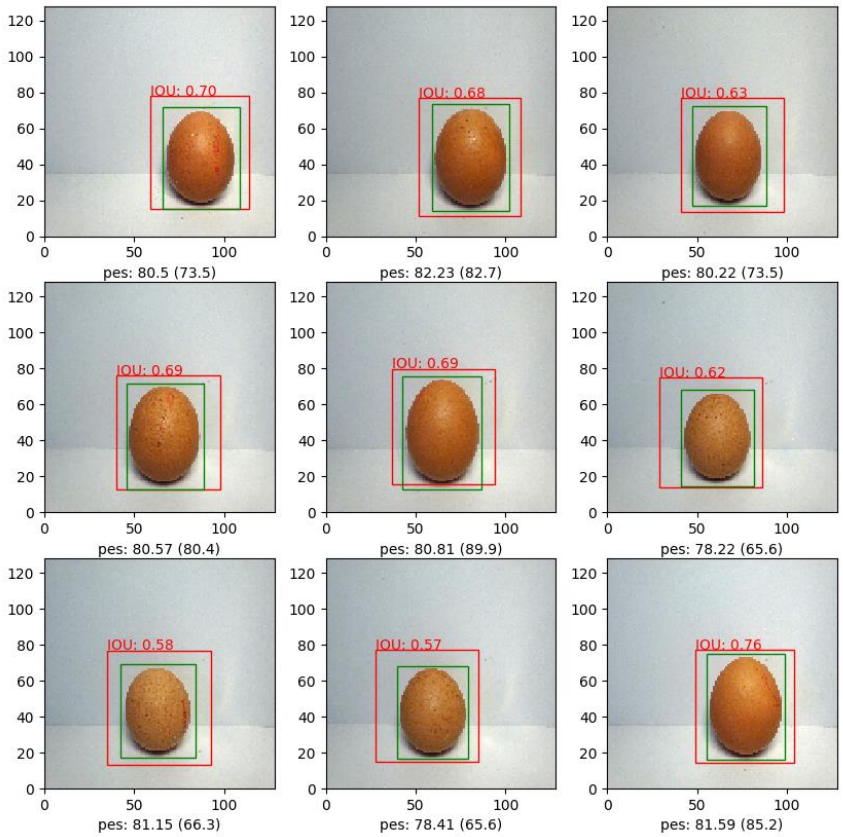

Figura 8.7. Mostra d'algunes prediccions del model pel *dataset* d'ous sense referència

[6] Error absolut mitjà $= \sum \frac{|\text{ valor correcte} - \text{valor predit }|}{n}$

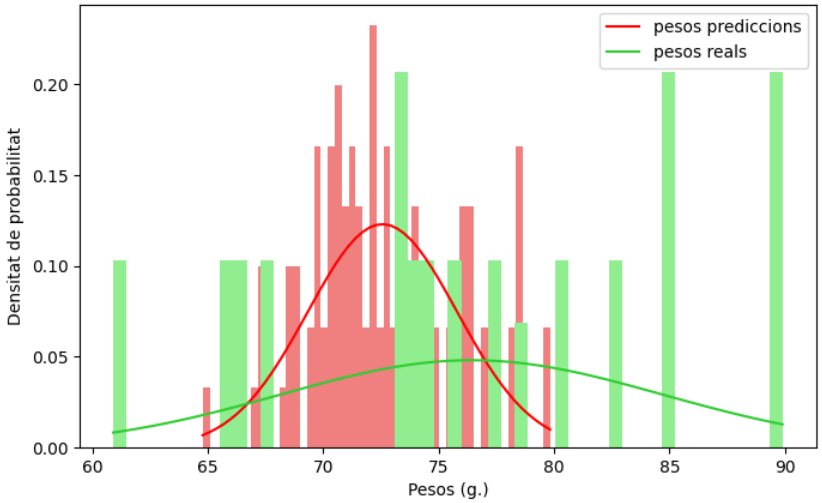

Figura 8.8. Distribucions dels pesos reals (en verd) en comparació amb els predits (en vermell).

	Distribució real	**Distribució predita**
Mitjana del pes (g.)	76,43	72,57
Desviació típica	8,28	3,24

Taula 8.2. Mitjanes i desviacions típiques de les distribucions i IoU mitjà de la predicció

IoU mitjà de la predicció = 0,66

Error absolut mitjà del pes = 6,04 g

8.2.2 Model 2: entrenat sense referència

Igual que l'anterior model, aquest estat entrenat amb 5000 imatges *augmented*, generades a partir de 250 imatges (de les quals només contenen 50 ous diferents). A diferència de l'anterior model, aquestes imatges no contenen cap objecte de referència.

A continuació, aquest *dataset* de 5000 imatges s'ha separat amb imatges de *training* i de *validation* i s'ha dut a terme l'entrenament del model. Amb 50 iteracions les gràfiques per cadascuna de les sortides són les següents:

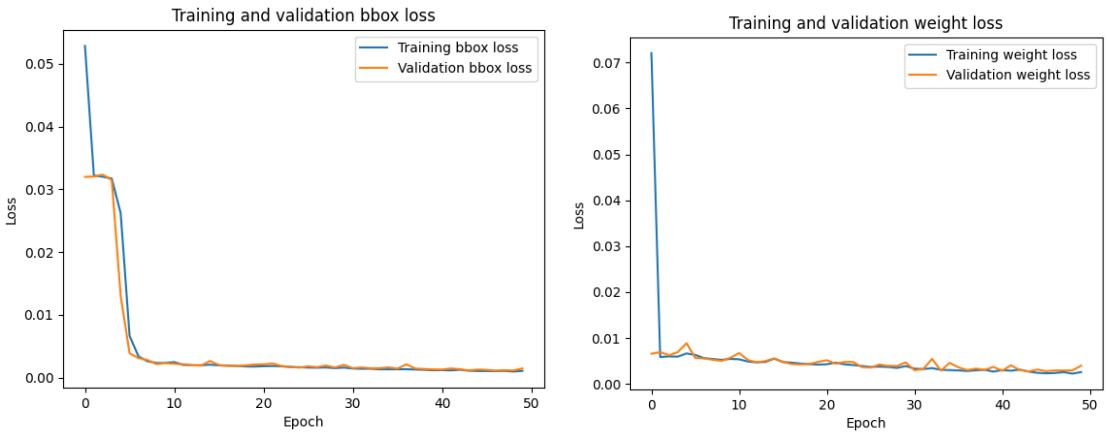

Figura 8.9. Gràfica de l'error de *training* i *validation* tant per la sortida de la *bounding box* (esquerra) com pel pes (dreta)

Inicialment l'error és molt gran ja que els pesos del model s'inicialitzen aleatòriament, però a mesura que van passant les iteracions es van ajustant als valors òptims.

Com en l'anterior cas, es testejarà el model amb dos *datasets* diferents, un amb imatges d'ous amb referència i un altre amb imatges amb ous sense referència.

Test amb *dataset* d'ous amb referència

Aquest *dataset* està format per 50 fotos d'ous amb referència, cap de les quals ha servit ni per generar *augmented images* ni per avaluar el model. A continuació es mostraran els resultats de les prediccions:

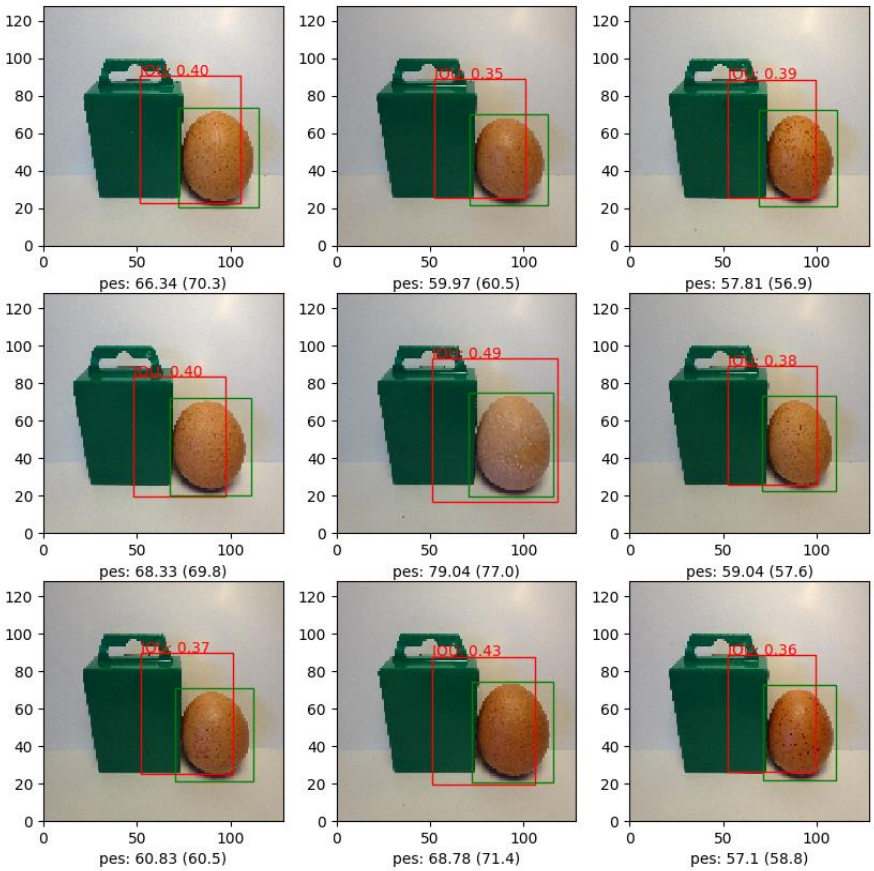

Figura 8.10. Mostra d'algunes prediccions. La *bounding box* real (en verd), la predita amb la mesura de l'IoU (en vermell) i a sota de cada imatge el pes predit i el pes real (aquest últim entre parèntesis).

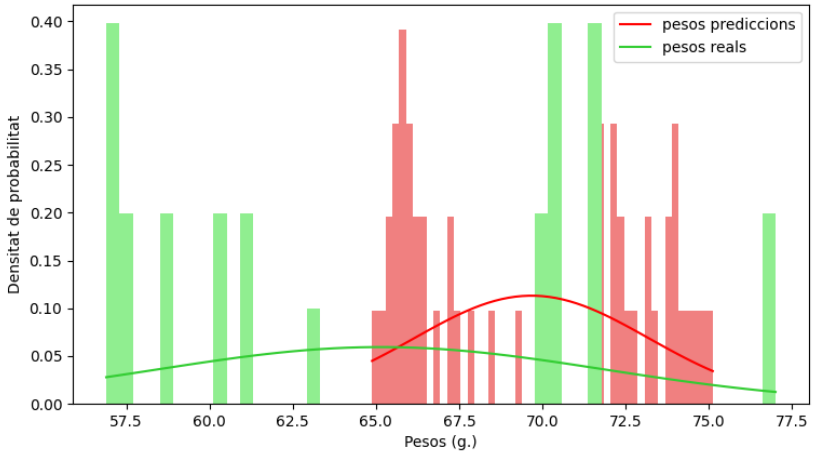

Figura 8.11. Distribucions dels pesos reals (en verd) en comparació amb els predits (en vermell).

	Pesos reals	Predicció
Mitjana del pes (g.)	65,15	69,67
Desviació típica	6,71	3,53

Taula 8.3. Mitjanes i desviacions típiques de les distribucions i IoU mitjà de la predicció

IoU mitjà de la predicció = 0,40

Error absolut mitjà del pes = 4,91 g

Test amb *dataset* d'ous sense referència

Aquest *dataset* està format per 50 fotos d'ous sense referència, cap de les quals ha servit ni per generar *augmented images* ni per avaluar el model. A continuació es mostraran els resultats de les prediccions:

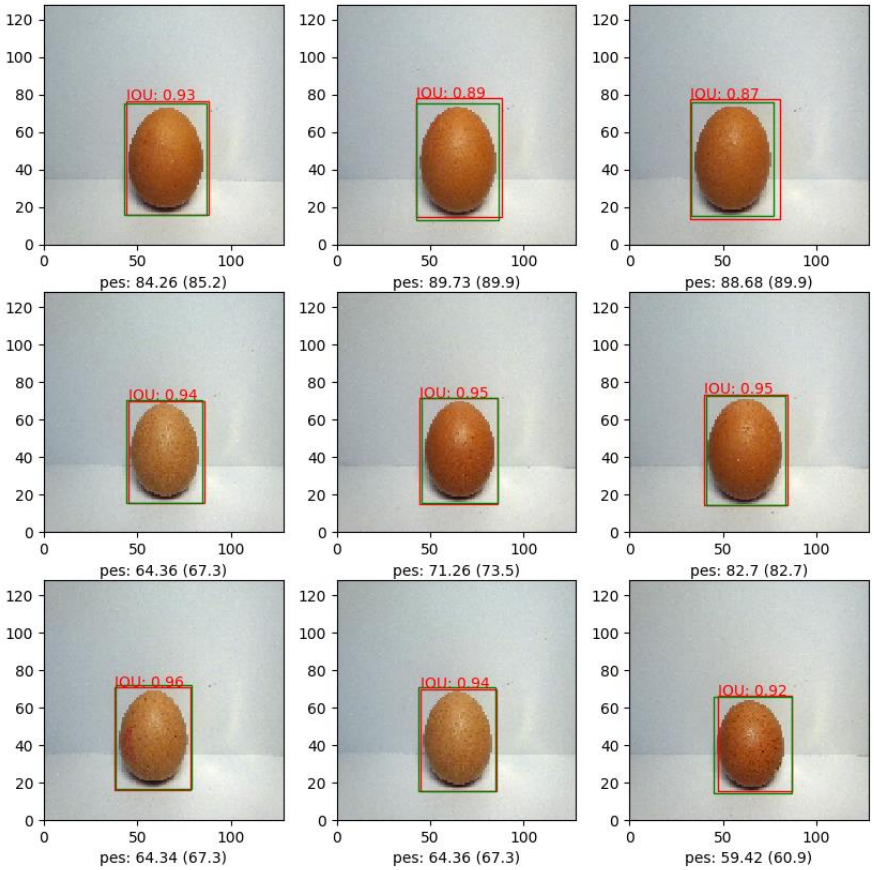

Figura 8.12. Mostra d'algunes prediccions

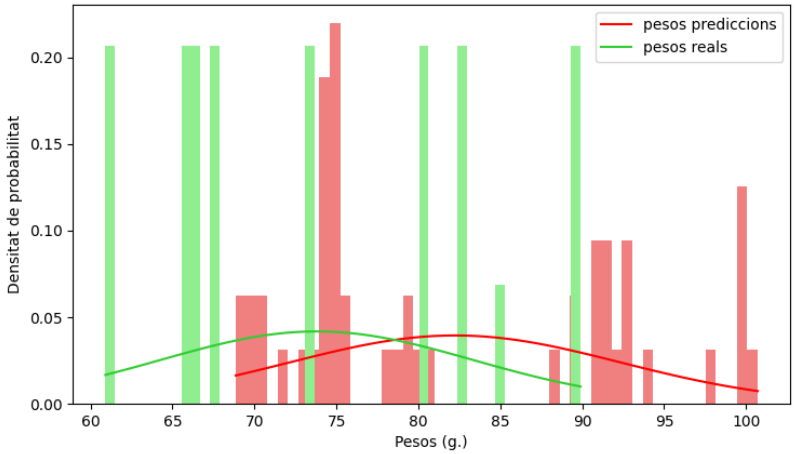

Figura 8.13. Distribucions dels pesos reals (en verd) en comparació amb els predits (en vermell).

	Pesos reals	**Predicció**
Mitjana del pes (g.)	73,8	82,24
Desviació típica	9,53	10,09

Taula 8.4. Mitjanes i desviacions típiques de les distribucions i IoU mitjà de la predicció

IoU mitjà de la predicció = 0,91

Error absolut mitjà del pes = 8,44 g

8.3 Comparativa dels dos models

Un cop entrenats i testejats els dos models, es pot observar que la informació de la referència ajuda a la xarxa a predir el pes.

També observem que, com era d'esperar, cadascun dels models obté millors resultats avaluant-ho amb les imatges que s'ha entrenat (és a dir, el model entrenat amb referència té més precisió avaluant el dataset de test amb ous amb referència que no l'altre. El mateix passa amb l'altre model).

En un principi s'esperava que el model entrenat amb referència tingués bons resultats en ambdós tests, mentre que el model entrenat sense referència, al avaluar-lo amb ous amb referència, tingués molt d'error (ja que no reconeixeria la caixa verda). Tot i així, en els resultats s'observa que l'error no és excessivament gran; un dels motius pot ser que,

tot i que els ous de test siguin diferents als d'entrenament, les condicions de l'entorn variïn poc (cosa que no passa amb l'altre model)

També es pot veure que la precisió de la *bounding box* (IoU) està relacionada amb la precisió dels pesos: això és degut a que les dues *loss functions* s'ajuden a l'hora d'optimitzar, i funciona millor envers si els models de identificació de l'ou i predicció del pes anessin per separat. El fet de tenir dues sortides de la mateixa xarxa és un avantatge.

Cal comentar que evidentment això s'ha fet amb *datasets* petits, i per tal que els resultats fossin més reals s'hauria de testejar amb moltes més imatges. En aquest cas, però, al tenir unes imatges limitades s'ha optat per fer-ne servir més per entrenar i unes poques per testejar.

9. Planificació

En aquest apartat s'explica la planificació seguida a l'hora de confeccionar el projecte, detallant-ne les diferents fases i el temps que ha suposat cadascuna d'elles.

Fase 1: aprenentatge i estudi del problema

La primera fase va ser d'aprenentatge del *Deep Learning*. Es van mirar cursos online, es van comprar llibres, es va practicar amb exemples: en resum, es va fer molta recerca per tal d'assolir els conceptes bàsics d'aquest camp. També en aquesta fase es va decidir el problema concret d'estudi i es van buscar treballs relacionats.

Fase 2: implementació i proves

En aquesta fase, ja amb una mica més de noció de *Deep Learning*, es va començar tractar el problema en qüestió i es va començar a passar a la pràctica. Es van fer fotos a ous, es van etiquetar i es van tractar les dades per tal de realitzar les proves inicials. Més endavant, es va desenvolupar un primer model capaç de només predir el pes. A continuació un altre model, per separat, capaç només de només identificar l'ou. Un cop semblava que aquests ja començaven a funcionar, es va implementar el *data augmentation*, amb l'objectiu d'augmentar les dades d'entrada i així millorar l'entrenament del model.

Val a dir que aquí es seguien consultant recursos i es seguia fent recerca de *Deep Learning*, ja que a mesura que s'anava avançant també era quan anaven apareixent nous problemes.

Fase 3: validació del model final i creació Web App

En aquesta fase es van ajuntar els dos models anteriors en un de sol, capaç de dur a terme les dues tasques alhora (identificació de l'ou per una banda i regressió del pes per l'altra). Es van acabar de retocar alguns paràmetres de l'estructura de la xarxa i finalment es va donar el model per bo (resultats suficients). A partir d'aquí, es va començar a dissenyar la visualització de la interfície gràfica (amb un aprenentatge previ de les eines a utilitzar), fins a la creació de la *Web App*.

Fase 4: redacció memòria

Finalment, amb tota la part pràctica finalitzada, es va fer la redacció de la memòria.

En el diagrama de Gantt següent es detallen les dates aproximades de cada part del projecte.

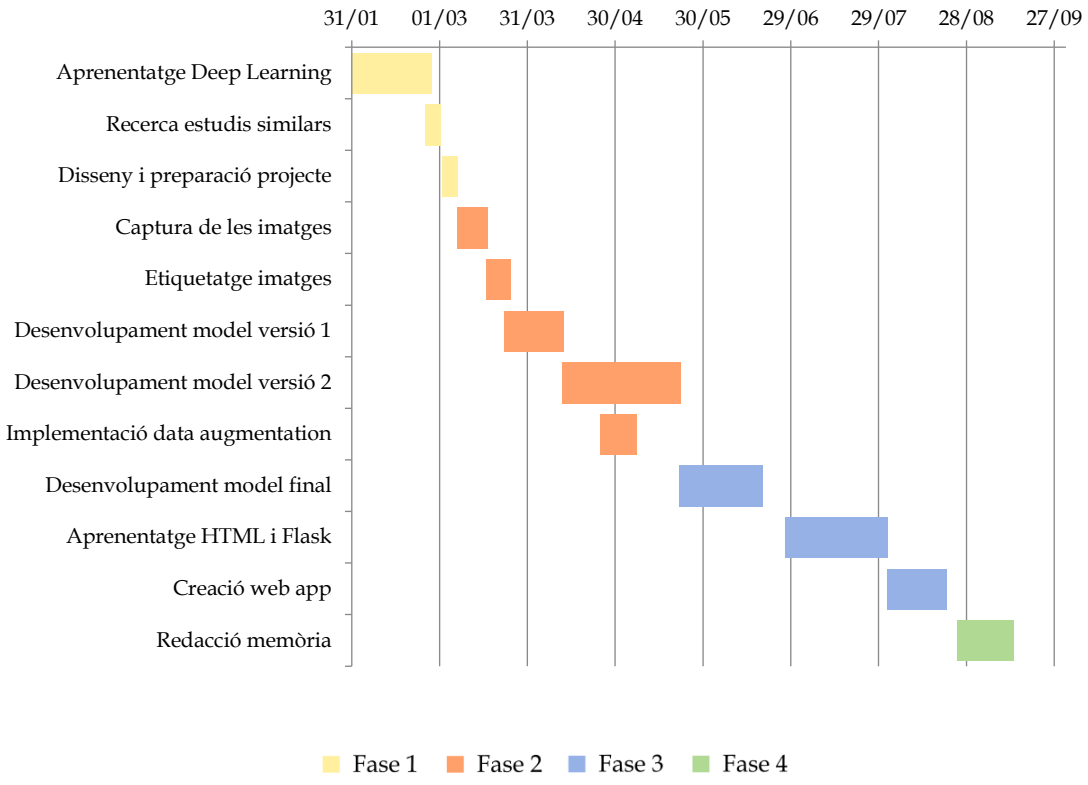

Figura 9.1. Diagrama de Gantt de la planificació del projecte

10. Costos

Els costos associats al projecte es poden dividir entre els costos de mà d'obra i els costos de maquinària.

Costos mà d'obra

Aquest treball s'ha elaborat durant 7 mesos, dedicant-hi una mitjana de 2,5 h/dia (es consideren només dies laborables), de manera que fa un total de 350 hores dedicades. Si considerem que el preu/h d'un enginyer de software és de 25€/h, això ens surt un total de:

$$560\ hores\ \cdot 25\frac{€}{hora} = 8.750€ \tag{10.1}$$

Costos de maquinària i productes

L'eina principal d'aquest treball ha estat l'ordinador portàtil, un MSI GF75 Thin utilitzat pel tractament d'imatges, la programació del codi i la redacció del treball. El seu preu és de 1.099 € amb una vida útil estimada de 10 anys, de manera que en aquest treball ha suposat un cost de 64,11 € (considerant que s'ha utilitzat els 7 mesos que ha durat el treball).

També es va utilitzar una càmera digital Sony DSC-WX350 per fer les fotos als ous. Aquesta costa 280 €, i si suposem que també dura uns 10 anys ens surt un cost de 4,67 € (com que el *dataset* es va fer al principi del treball, s'ha considerat 2 mesos d'utilització de la càmera).

Per pesar els ous es va comprar una balança digital Uzinb de 5,92 €; en aquest cas com que sí es va comprar expressament pel treball es computarà tot el preu d'aquesta en els costos.

Per la captura de les fotos dels ous es va construir un panell amb planxes d'aglomerat per tal que el fons de les imatges sempre fos el mateix (color blanc) i poder tenir un major control sobre els factors de l'entorn de les imatges (lluminositat, distància, angle de la càmera, etc.).

Figura 10.1. Panell utilitzar per capturar imatges dels ous

Així doncs, per a la construcció d'aquest es va fer servir:

- 2 x Planxa d'aglomerat blanc (5,49 €/unitat)
- 2 x Escaire blanc (3,95€/unitat)
- Cargols per fusta (2,95€ un paquet)

Finalment, el producte fonamental del treball: els ous de gallina. En total es van comprar 10 paquets de 6 ous. Com eren de marques i mides d'ous diferents (i així s'obtenia més variabilitat), pel càlcul dels costos s'ha considerat que el preu mitjà d'un paquet de mitja dotzena d'ous és de 1,5 €. Així doncs, en total representa un cost de 15 €.

Els costos de maquinària queden resumits en la següent taula:

Producte	**Cost en el treball (€)**
Portàtil MSI GF75 Thin	64,11
Càmera Sony DSC-WX350	4,67
Balança digital Uzinb	5,92
Panell per les fotos	21,83
Ous de gallina	15
Total	111,53

Taula 10.1. Desglossament del cost de maquinària

Cost total

Així doncs, el cost total del projecte és de:

Costos mà d'obra	8.750 €
Costos maquinària	111,53 €
Cost total	**8.861,53 €**

Taula 10.2. Resum del cost total del projecte

Conclusions

Per acabar, es farà un breu resum de les diferents fases del projecte, amb els problemes que s'han anat trobant en cadascuna d'elles, les solucions proposades i les conclusions extretes.

El primer problema que es va trobar va ser que per entrenar una xarxa neuronal calia disposar d'un *dataset* suficientment gran, i això implicava moltes imatges. Per no haver de comprar una quantitat enorme d'ous ni perdre molt temps, el que es va fer va ser, per una banda, capturar més d'una foto (amb petits canvis) i, per l'altra, es va fer *data augmentation* de les imatges capturades. Amb aquestes dues tècniques es va aconseguir generar 5.000 imatges per l'entrenament de la xarxa a partir de només 60 ous reals.

El segon problema es trobava en el desenvolupament del model que havia de dur a terme les prediccions. Després d'un anàlisi previ, es va decidir crear una xarxa neuronal convolucional (implementada des de zero) capaç de treure dues sortides (localització i pes de l'ou) a partir d'una única imatge d'entrada. Probablement en aquesta part és on rau la quantitat més gran de feina del treball, ja que el procés d'ajustament dels paràmetres pel correcte aprenentatge va implicar molt de temps. Un cop avaluat el model, es fa evident la importància de les dades amb què s'entrena, i s'arriba a la conclusió que per a tasques de regressió del pes d'un objecte a partir d'imatges 2D, el model funciona millor si disposa d'una referència. Val a dir que tot i no assolir uns resultats excepcionals en les prediccions, es demostra el gran potencial d'aquestes xarxes en l'anàlisi d'imatges i la gran utilitat del *Deep Learning* en tasques industrials.

Finalment, per solucionar el problema de la visualització dels resultats i per tal d'apropar l'estudi a altres usuaris, es va crear una interfície gràfica (mitjançant una aplicació web) que permet carregar imatges d'ous de gallina i fer prediccions de la localització i el pes d'aquests.

Treballs futurs

En aquest apartat es comentaran algunes de les millores que es podrien aplicar a aquest treball per tal d'ampliar-lo i perfeccionar-lo.

Dataset

Una de les limitacions alhora d'entrenar el model és la quantitat d'ous de gallina reals utilitzats. Tot i haver generat noves imatges a partir del *data augmentation*, si es volgués que aquest projecte tingués millors resultats s'haurien de fer moltes més fotos a ous reals (permetent així que el model veiés més tipus d'ous). Una possible solució seria anar a una granja de producció d'ous de gallina i capturar moltes imatges. Amb això no significa que llavors no calgués fer *data augmentation* (en *Deep Learning* gairebé sempre se'n fa), però sí que permetria que el model generalitzés millor.

Una altra possible millora que es podria introduir és en la fase de fer la foto i registrar el pes. Ara mateix primer es pesava l'ou, s'anotava manualment i llavors es feia la foto amb una càmera; una solució seria tenir un sistema integrat que ho fes tot alhora, i evitar així possibles errors.

Xarxa neuronal

Pel que fa a la xarxa neuronal, es podria estendre en tant que, a més de predir-ne la localització i el pes, fos capaç de detectar defectes (com ara esquerdes o brutícia en la clova de l'ou).

Una altra possible millora, com feien alguns estudis en l'apartat 4.1, seria canviar l'actual mètode de predicció del pes. Una manera seria anotar diversos punt de l'ou sobre la imatge i fer una aproximació del volum a partir d'aquesta informació. O bé també es podria introduir més d'una imatge de l'ou (per exemple una frontal, una lateral i una superior), calcular-ne el volum i a partir d'aquí el pes (evidentment això implicaria tenir més càmeres i per tant el pressupost també augmentaria).

En el camp del *Deep Learning* la potència de computació és un aspecte bàsic. En aquest treball únicament s'ha fet servir el meu ordinador portàtil, però per tal d'agilitzar el procés d'entrenament es podria fer ús d'un servidor extern més potent.

Interfície gràfica

Un dels aspectes més importants a millorar en la interfície gràfica és el temps de predicció (des que l'usuari carrega una imatge fins que obté els resultats). Una solució seria que el sistema no necessités guardar la imatge nova creada (amb la *bounding box*

dibuixada), sinó que fes la predicció i mostrés els resultats sempre treballant només amb la imatge d'entrada.

També es podrien mostrar algunes estadístiques en la pantalla de resultats referents a la predicció, com ara la precisió d'aquest o bé el temps trigat en a calcular-ho.

Finalment, per tal que aquest sistema fos més real i pogués ser utilitzat en un entorn de producció, hauria de poder treballar en temps real. Així doncs, en la interfície gràfica hi podria haver una modalitat de treball "En directe" que es connectés a la càmera de l'ordinador (o a una càmera USB) i fes prediccions al mateix moment que la càmera capta les imatges.

Agraïments

M'agradaria agrair a totes aquelles persones que han estat al meu costat durant els darrers anys, els amics, i en especial a la meva família i a la meva parella, que sempre m'han donat suport i han estat al meu costat pel que fes falta.

Donar les gràcies també al meu tutor, Lluís Solano, per la disponibilitat constant i les indicacions donades en les diferents fases del treball.

Agrair també al meu cosí, Pau Bramon, el consell i guiatge en la realització d'algunes parts del projecte.

Bibliografia

[1] J. &. N. P. K. &. T. K. &. C. T. &. L. W. Yeow, «Effects of Stress, Repetition, Fatigue and Work Environment on Human Error in Manufacturing Industries.,» *Journal of Applied Sciences,* 2014.

[2] C. C. a. P. S., «Object weight estimation from 2D images,» *ARPN Journal of Engineering and Applied Sciences,* 2015.

[3] K. K. S. Radhamadhab Dalai, «Weight Estimation through Image Analysis,» *International Journal of Computer Trends and Technology (IJCTT),* 2017.

[4] M. S. a. J. K. J. Lehr, «Classification of Similar Objects of Different Sizes Using a Reference Object by Means of Convolutional Neural Networks,» *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA),* September 2019.

[5] J. Nelsoon, «Getting started with labelimg for labelling object detection data,» Març 2020. [En línia]. Available: https://blog.roboflow.com/getting-started-with-labelimg-for-labeling-object-detection-data/. [Últim accés: Abril 2020].

[6] P. developers, «Pandas. Python Data Analysis Library,» Setembre 2020. [En línia]. Available: https://pandas.pydata.org/.

[7] J. Torres, Python Deep Learning, Barcelona: Marcombo, 2020.

[8] J. I. Bagnato, «Aprende Machine Learning,» Novembre 2018. [En línia]. Available: https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/. [Últim accés: Setembre 2020].

[9] V. Bushaev, «Adam Optimizer,» Octubre 2018. [En línia]. Available: https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c. [Últim accés: Maig 2020].

[10] Google, «Tensorflow. An Open Source Machine Learning Framework for Everyone,» 2020. [En línia]. Available: https://www.tensorflow.org/?hl=es-419. [Últim accés: Març 2020].

[11] A. Jung, «Imgaug. Image augmentation for machine learning,» 2020. [En línia]. Available: https://imgaug.readthedocs.io/en/latest/. [Últim accés: Juny 2020].

[12] A. Ronacher, «Flask,» [En línia]. Available: https://flask.palletsprojects.com/en/1.1.x/.

[13] J. D. Hunter, «Matplotlib. Python plotting,» 2016. [En línia]. Available: https://matplotlib.org/. [Últim accés: Abril 2020].

[14] A. Rosebrock, «PyImage Search,» Novembre 2016. [En línia]. Available: https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/. [Últim accés: Setembre 2020].

[15] A. Rosebrock, «PyImageSearch. Keras, Regression, and CNNs,» Gener 2019. [En línia]. Available: https://www.pyimagesearch.com/2019/01/28/keras-regression-and-cnns/. [Últim accés: Març 2020].

[16] J. Rieke, «Object detection with neural networks — a simple tutorial using keras,» Juny 2017. [En línia]. Available: https://towardsdatascience.com/object-detection-with-neural-networks-a4e2c46b4491. [Últim accés: Maig 2020].

[17] A. Rosebrock, «PyImageSerach. Keras ImageDataGenerator and Data Augmentation,» Juliol 2019. [En línia]. Available: https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/. [Últim accés: Maig 2020].