Treball de Fi de Grau

Grau en Enginyeria en Tecnologies Industrials (GETI)

SISTEMA DE CALIBRATGE DE PEIXOS

MEMÒRIA

Autor: Guillem Costa Gabaldón

Director: Vicenç Parisi

Convocatòria: Juny 2021

Escola Tècnica Superior

d'Enginyeria Industrial de Barcelona

Resum

Aquest projecte es centra en el de desenvolupament d'un sistema de segmentació semàntica per imatges de capses plenes de pops blancs provinents de les llotges de Llançà i Palamós. Per la seva realització s'ha utilitzat la xarxa neuronal convolucional Mask R-CNN i les llibreries de programació Tensorflow i Keras.

El treball de final de grau en qüestió és part d'un projecte realitzat conjuntament amb altres estudiants per oferir solucions de realitat augmentada a les llotges de pescadors. L'objectiu final és fer un detector que pugui reconèixer totes les espècies que es venen a les llotges.

En aquest document es troba la metodologia seguida per desenvolupar un detector de contorns amb l'objectiu de detectar el total de pops blancs en cada capsa. Partint del repositori de GitHub, *Mask R-CNN for Object Detection and Segmentation* [1], s'han modificat paràmetres i funcions i creat nous programes per tal d'adaptar aquesta implementació al mateix projecte. També s'ha creat una base de dades formada per 410 imatges etiquetades a mà. Els entrenaments i els programes d'inferència s'han executat en Google Colaboratory, que ha permès accelerar aquests processos.

Els resultats obtinguts demostren una bona precisió en les segmentacions dels contorns dels pops blancs. L'error mitjà en el nombre d'exemplars que es detecten per imatge és d'1,63 pops blancs. Tot i això, el model podria millorar amb una base de dades més gran. També s'ha intentat predir el calibre dels pops blancs a partir de les àrees d'aquests, però els resultats obtinguts no han permès aquesta aplicació final.

Índex

1 Glossari 7

2 Introducció 9

2.1 Problemàtica 9

2.2 Objectius del projecte 9

2.3 Abast del projecte 9

2.4 Estat de l'art 10

3 Marc Teòric 13

3.1 Visió per computador 13

3.2 Machine learning 14

3.3 Deep learning 17

3.4 Xarxes Neuronals Artificials 17

3.4.1 Funció d'activació 19

3.4.2 Entrenament 21

3.4.3 Funció de pèrdues 22

3.4.4 SGD 22

3.4.5 Bias 24

3.5 Xarxes neuronals convolucionals (CNN) 25

3.6 Mask R-CNN 31

3.7 Tensorflow i Keras 38

4 Creació de la base de dades i entrenament del sistema 39

4.1 Obtenció de les imatges 39

4.2 Preparació de les imatges 39

4.3 Entorn per realitzar l'entrenament i la inferència 41

4.4 Divisió de la base de dades 42

4.5 Entrenament 42

5 Anàlisi de resultats 43

5.1 Entrenament 0 43

5.2 Entrenament 1 47

5.3 Entrenament 2 48

5.4 Entrenament 3 50
5.5 Entrenament 4 52
5.6 Estudi de les àrees i classificació del calibre 57
5.7 Creació del programa d'inferència 63
6 Planificació temporal 65
7 Estudi econòmic 67
8 Impacte ambiental 69
Conclusions 71
Agraïments 73
Bibliografia 75
Referències bibliogràfiques 75

Índex de figures

Figura 1: Diferents maneres de detectar objectes a partir de visió artificial 14
Figura 2: Exemple d'aplicació d'algoritmes de clustering en la base de dades wine dataset 15
Figura 3: Exemple d'ús d'autoencoders en la base de dades MNIST 16
Figura 4: Exemple d'una xarxa neuronal artificial 18
Figura 5: Corba de la funció d'activació ReLu 19
Figura 6: Corba de la funció d'activació sigmoid 20
Figura 7: Procés realitzat pel SGD 23
Figura 8: Procés realitzat pel SGD en 3 models amb diferent learning rates 23
Figura 9: Xarxa neuronal artificial 24
Figura 10: Exemple d'operació convolucional d'una imatge 6 x 6 x 3 amb un filtre 3 x 3 27
Figura 11: Exemple de productes convolucionals 28
Figura 12: Resultat d'una capa max pooling amb filtre 2 x 2 i stride de 2 29
Figura 13: Exemple CNN 31
Figura 14: Output Mask R-CNN 32
Figura 15: Centres dels anchors en una imatge 33
Figura 16: Imatge abans i després d'aplicar NMS 34

Figura 17: Funcionament del pas 3 per una proposta de bounding box 35
Figura 18: Arquitectura R-CNN 36
Figura 19: Funcionament Mask R-CNN 37
Figura 20: Imatge d'una capsa de pops blancs de la subhasta de la llotja de Llançà 39
Figura 21: Capsa ordenada (esquerra) i desordenada (dreta) 40
Figura 22: Edició d'una imatge amb labelme 41
Figura 23: Màscares fetes amb labelme (esquerra) i obtingudes amb el model de l'entrenament 0 (dreta) 44
Figura 24: Màscares amb labelme (esquerra) i obtingudes de l'entrenament 0 (dreta) 44
Figura 25: Pèrdues de l'entrenament 0 per 100 epoch (esquerra) i per 200 epoch (dreta) 45
Figura 26: Corbes de pèrdues de quatre models diferents 46
Figura 27: Pèrdues entrenament 0 (esquerra) i entrenament 1 (dreta) 47
Figura 28: Màscares obtingudes del model de l'entrenament 0 (esquerra) i 1 (dreta) 49
Figura 29: Màscares obtingudes del model de l'entrenament 2 (esquerra) i dibuixades a mà amb el labelme (dreta) 49
Figura 30: Pèrdues entrenament 1 (esquerra) i entrenament 2 (dreta) 50
Figura 31: Pèrdues entrenament 3 51
Figura 32: Pèrdues d'entrenament (esquerra) i de validació (dreta) 52
Figura 33: Imatge original (esquerra) i màscares obtingudes pel model (dreta) 54
Figura 34: Imatge original (esquerra) i màscares obtingudes pel model (dreta) 54
Figura 35: Imatge original (esquerra) i màscares obtingudes pel model (dreta) 55
Figura 36: Imatge original (esquerra) i màscares obtingudes pel model (dreta) 55
Figura 37: Imatge original (esquerra) i màscares obtingudes pel model (dreta) 56
Figura 38: Imatge original (esquerra) i màscares obtingudes pel model (dreta) 56
Figura 39: Histograma d'àrees 57
Figura 40: Histograma d'àrees classificat per calibres 58
Figura 41: Diagrama de caixes d'àrees classificat per calibre 58
Figura 42: Diagrama de caixes d'àrees classificat per calibres 59
Figura 43: Principals components i variància (PCA) 60
Figura 44: Capsa de pops blancs amb màscara 61
Figura 45: Màscara editada amb skeletonize (bon resultat) 62

Figura 46: Màscara editada amb skeletonize (resultat dolent) 62
Figura 47: Output de la funció Pop_blancInf 64
Figura 48: Diagrama de Gantt del projecte 65

Índex de taules

Taula 1: Error mitjà amb els pesos de l'entrenament 3 amb 381, 386 i 400 epoch 51
Taula 2: Error en el nombre de pops i pes, per imatges de test de l'entrenament 4. 53
Taula 3: Error de pops/kg per imatges de test de l'entrenament 4. 53
Taula 4: Mitjana de l'error absolut per imatges de test i train de l'entrenament 4. 57
Taula 5: Influència de l'àrea i el nombre de pops en els components principals 60
Taula 6: Resultats classificació amb Neraest Neighbors 61
Taula 7: Costos del projecte 68

1 Glossari

GPU: *Graphics Processing Unit.*

CNN: *Convolutional Neural Network.*

Epoch: Quan tota la base de dades passa per la xarxa neuronal artificial.

Bounding box: Rectangle que indica la ubicació d'un objecte en una imatge.

Etiqueta: Informació significativa sobre una dada de la base de dades que li permet al model aprendre.

Model: Xarxa neuronal que ha estat entrenada amb una base de dades.

2 Introducció

2.1 Problemàtica

Actualment, els compradors de les llotges de peix de Catalunya disposen de pocs segons per determinar si volen comprar una capsa de peix. Seria de gran ajuda, doncs, oferir una eina que proveís més informació als compradors de la caixa en qüestió per tal de facilitar la seva decisió. Aquest projecte intenta fer un programa que amb l'ajuda d'una xarxa neuronal convolucional detecti el nombre d'espècies que hi ha en la capsa i les seves dimensions. L'espècie en concret serà el pop blanc i es farà l'estudi a partir de les fotografies de les capses de les llotges de Llançà i Palamós del febrer i març de 2021.

2.2 Objectius del projecte

El principal objectiu d'aquest projecte és crear un programa capaç d'identificar el nombre total de pops blancs en una imatge. Concretament detectarà els caps dels pops blancs. A més a més, s'intentarà fer un estudi de les àrees i de la longitud dels pops de les imatges per intentar predir el calibre d'aquests. El calibre és una dada que ens proporcionen les llotges i fa referència a la mida dels pops blancs de la caixa.

Com a futur objectiu es podria utilitzar aquest programa per millorar el sistema de subhasta que tenen actualment a les llotges i implementar-ho per cada espècie.

2.3 Abast del projecte

La creació de la base de dades, l'entrenament i l'anàlisi de resultats per les fotografies de capses amb pops blancs. També s'inclou dins l'abast del projecte la creació de diferents programes per fer la inferència de resultats i la comprensió del funcionament de la xarxa neuronal convolucional Mask R-CNN. Dintre de l'abast no s'inclou la implementació del programa en les llotges.

2.4 Estat de l'art

En l'última dècada, el camp de la visió artificial i l'aprenentatge automàtic ha avançat molt sobretot gràcies a les xarxes neuronals convolucionals (CNN). Però aquest no és un camp nou, sinó que durant aproximadament seixanta anys, els científics de la computació de tot el món han estat tractant de trobar formes de fer que les màquines trobessin significats de les dades visuals.

A continuació es farà un breu resum dels avenços de la visió artificial, l'aprenentatge automàtic i l'aprenentatge profund [2]:

David Hubel i Torsten Wiesel, el 1959 van publicar un dels articles que més ha influït en el camp de l'aprenentatge automàtic. En la seva publicació anomenada *Receptive fields of single neurones in the cat's striate cortex* [3], exposen com accidentalment van descobrir quin estímul visual produïa que s'excités una neurona en el cervell d'un gat. Aquest estímul va ser la línia creada per l'ombra de la vora d'un objecte. A partir d'això, van establir que hi ha neurones simples i complexes en l'escorça visual primària i que el processament visual sempre comença amb estructures simples com vores. Aquest és el principi fonamental de l'aprenentatge profund.

En el 1959, Russell Kirsch i els seus companys van crear un aparell que permetia convertir imatges en matrius de números que les màquines amb llenguatge binari podien entendre. Gràcies a això ara podem processar imatges digitals de diverses formes.

El 1982, David Marr va publicar l'article *Vision: A computational investigation into the human representation and processing of visual information* [4]. Basant-se en la idea de David Hubel i Torsten Wiesel, Marr exposa que la visió és jeràrquica. Segons ell, les neurones detecten característiques simples com vores, després s'alimenten de característiques més complexes com a formes i finalment, s'alimenten de representacions visuals més complexes.

En el mateix any, Kunihiko Fukushima, inspirat també pel treball de David Hubel i Torsten Wiesel va crear una xarxa artificial convolucional formada per cèl·lules simples i complexes que podien reconèixer patrons i no es veien afectades pels canvis de posició. Aquesta xarxa es va anomenar Neocognitron i incloïa diverses capes convolucionals.

Uns anys més tard, el 1989, Yann LeCun es va basar en la Neocognitron per crear una altra xarxa neuronal convolucional anomenada LeNet-5, que utilitzava un algoritme d'aprenentatge anomenat *backpropagation,* el qual actualment s'utilitza en moltes xarxes neuronals. Així mateix, Yann LeCun també va crear la base de dades MNIST, que esdevé la més coneguda dins del camp de l'aprenentatge automàtic.

Paul Viola i Michael Jones el 2001 van presentar el primer detector de rostres que funcionava en temps real. De la mateixa manera que en les xarxes neuronals, mentre l'algoritme processava les imatges, aquest detector era capaç d'aprendre quines característiques podien ajudar a localitzar cares.

En el 2009, Pere Felzenszwalb, David McAllester i Deva Ramanan van desenvolupar un altre detector important, el model de peça deformable (DPM). Aquest va mostrar un gran rendiment en les tasques de detecció d'objectes, on es van utilitzar quadres delimitadors per localitzar-los.

El 2010, es va fer la primera edició del concurs ILSVRC [5] (*ImageNet Large Scale Visual Recognition Challenge*). Aquest concurs consisteix a fer un classificador d'imatges a gran escala utilitzant la base de dades ImageNet, que està formada per més d'un milió d'imatges amb més de mil classes (tipus d'objectes) diferents. En 2012 un equip de la Universitat de Toronto va participar amb un model de xarxa neuronal convolucional anomenat AlexNet que utilitzava una GPU. El model, similar en la seva arquitectura a LeNet-5 de Yann LeCun, va assolir una taxa d'error del 16,4%, que millorava en un 9,6% l'error de l'última edició. Era la primera vegada que es participava amb una CNN, fet que va suposar un moment decisiu per les CNN i la visó artificial.

En les següents edicions del concurs, l'ús de millors GPU va contribuir a disminuir l'error però molt breument, establint les xarxes neuronals convolucionals com a clares guanyadores des del 2012. Entre els guanyadors es troben la GoogleNet el 2014 i un any següent la Xarxa ResNet, la qual va aconseguir superar les capacitats humanes en la detecció d'objectes.

En el 2015 es va crear la U-Net, una CNN especialitzada en la segmentació d'imatges biomèdiques [6].

Com ja s'ha mencionat anteriorment, les CNN existeixen des de la dècada de 1980, però no s'han popularitzat fins fa deu anys a causa de la tecnologia d'aquella època. Actualment, gràcies al fet que els ordinadors són més potents, l'ús de GPU i l'accés a una gran quantitat de dades i imatges s'ha pogut avançar molt en el camp de la visió per computador.

A partir de l'èxit d'aquestes xarxes en el camp de la visió artificial, empreses com Google o Facebook han desenvolupat les llibreries TensorFlow i Pythorch respectivament. Actualment són les llibreries més utilitzades per la creació de xarxes neuronals artificials en el camp de l'aprenentatge profund.

En l'actualitat, es fa ús de la visió artificial per automatitzar molts processos on es necessitarien ulls humans. En el camp de la medicina per exemple, s'utilitza l'aprenentatge profund per detectar càncers de pell i de mama, tumors cerebrals o símptomes que s'estableixin com a patrons respiratoris per tal d'avaluar la progressió d'una persona malalta. En el camp de l'agricultura s'utilitza per detectar insectes pel control de plagues, controlar el reg o detectar la floració d'algunes espècies. Per últim, en el transport, algunes aplicacions de la visió per computador són: el reconeixement de matrícules, la detecció de vianants o sistemes per evitar col·lisions [7].

3 Marc Teòric

3.1 Visió per computador

La visió per computador és un cap científic multidisciplinari amb l'objectiu que els ordinadors puguin comprendre certes característiques d'una imatge de la mateixa manera que ho farien els humans. Una de les finalitats de la visió artificial és la detecció, reconeixement, localització i segmentació d'un objecte particular en una imatge.

Fins fa poc, la visió per ordinador només funcionava en una capacitat limitada, però gràcies als avenços en la intel·ligència artificial, el deep learning (aprenentatge profund), les xarxes neuronals artificials, el hardware i la quantitat d'imatges que es creen cada dia, en els últims anys, el camp de la visió per computador ha estat capaç d'avançar molt i actualment supera les capacitats dels éssers humans en algunes tasques com la de detecció i etiquetatge d'objectes.

Hi ha diversos nivells de com els ordinadors poden comprendre les imatges, anant d'una comprensió més general a una comprensió més específica [8]. A continuació, es procedeix a descriure'ls:

- **Classificació d'imatges:** Donada una imatge d'un objecte, l'orinador ha de ser capaç d'etiquetar-la, dient de quin objecte tracta. En la classificació d'imatges s'assumeix que només hi ha un objecte a classificar en la imatge.

- **Classificació amb localització:** Fa el mateix que en la classificació d'imatges, però a més, localitza on està l'objecte en la imatge. Això ho fa retornant les coordenades d'una *bounding box* (quadre delimitador) en la imatge.

- **Detecció d'objectes:** Les imatges ja no estan limitades a tenir només un objecte. La detecció d'objectes consisteix a classificar i localitzar (utilitzant *bounding boxes*) tots els objectes de la imatge.

- **Segmentació sistemàtica:** El seu objectiu és etiquetar cada píxel de la imatge. Assignant cada píxel a la classe corresponent del que s'està representant. A diferència dels anteriors, el resultat és una imatge d'alta definició on cada píxel és classificat en una classe diferent.

- **Segmentació d'instàncies:** Fa el mateix que la segmentació sistemàtica, però en aquest cas cada objecte el classifica en una classe diferent. En la figura 1 es pot veure clarament com en la segmentació sistemàtica només existeix la classe persona i en canvi en la segmentació d'instàncies els píxels corresponents a cada persona tenen una classe diferent.

En la figura 1 es pot veure un exemple dels diferents nivells que s'han explicat.

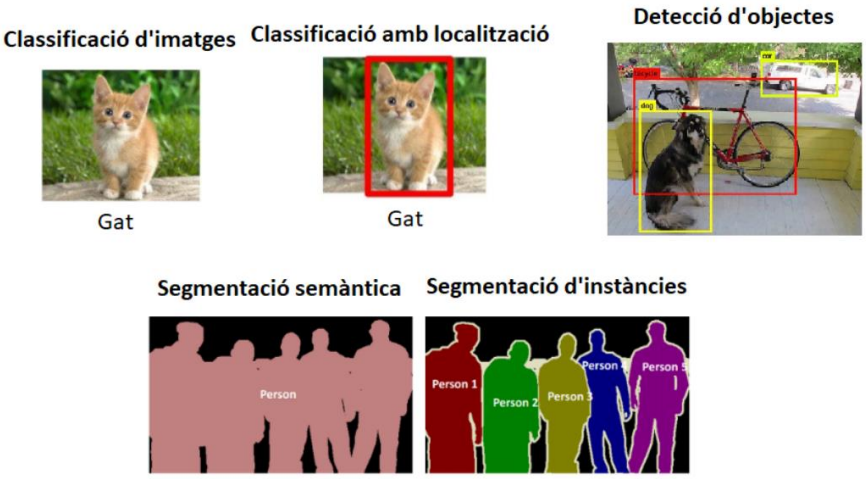


Figura 1: Diferents maneres de detectar objectes a partir de visió artificial

Font: Understanding Semantic Segmentation with UNET [9]

En aquest treball s'utilitzarà la segmentació sistemàtica per classificar tots els píxels corresponents al cap de pops blancs.

3.2 Machine learning

L'aprenentatge automàtic (en angles *machine learning*), és un camp de la intel·ligència artificial que està dedicat al disseny, l'anàlisi i el desenvolupament d'algorismes i tècniques que permeten analitzar dades i aprendre d'aquestes per després fer una determinació o predicció sobre noves dades a partir del reconeixement de patrons o classificació.

Aquesta definició recorda bastant a la definició de la programació clàssica, on s'escriu un algoritme i a continuació s'executa amb dades que el programa no ha vist abans per obtenir una resposta al problema. La principal diferència que hi ha entre l'aprenentatge automàtic i la programació clàssica està en el fet que l'aprenentatge automàtic li dóna molta importància a com aprèn de les dades. En comptes d'escriure manualment un conjunt d'instruccions per dur a terme una tasca específica, en l'aprenentatge automàtic la màquina experimenta un procés d'entrenament on s'introdueixen les dades juntament amb les respostes que s'espera d'aquestes i obté la capacitat d'executar la tasca sense dir-li específicament com fer-la [10].

Ficant un exemple, si es volgués fer un programa que classifiqués paraules en funció de si són verbs o noms, en la programació clàssica s'hauria d'indicar al programa tots els verbs i tots els noms que es volguessin classificar. I només podria classificar les paraules que se li haguessin dit. En canvi, en l'aprenentatge automàtic, el programa s'entrena amb moltes paraules i aprèn les característiques

que el fan ser un nom o un verb. Un cop ha après, el programa pot classificar paraules encara que mai les hagi vist.

En funció de les dades que se li atorguen al programa en la fase d'entrenament, existeixen diferents algoritmes per fer aprendre l'ordinador. A continuació s'explicaran tres dels més importants:

- **Aprenentatge supervisat:** És aquell en el que per entrenar el model se li dóna al programa les dades d'entrada i les respostes a aquestes dades. Direm que les dades estan etiquetades. L'aprenentatge supervisat és el que s'utilitzarà en aquest treball.

- **Aprenentatge no supervisat:** És aquell en què per entrenar el model se li dóna al programa només les dades d'entrada. En aquests models no es pot calcular la precisió, ja que no se sap la resposta. El model intentarà aprendre alguna estructura dintre de les dades d'entrada a partir d'identificar punts en comú i reaccionarà a la presència o absència d'aquests punts en les noves entrades. Una de les aplicacions més populars és l'ús d'algoritmes de *clustering*. Amb els mètodes que ja s'han comentat, el programa agruparà les entrades en grups o classes i una persona coneixedora de les dades haurà de determinar quina entrada correspon a cada grup. En la figura 2 es pot veure un exemple de l'ús de dos algoritmes de clustering, PCA i t-SNE en una base de dades, on cada entrada és un vi amb les seves característiques (àcid, alcohol, magnesi, etc.). El model ha estat capaç de crear aquesta figura a partir de les entrades i els seus atributs. Cada color fa referència a un tipus de vi diferent, 1,2 o 3.

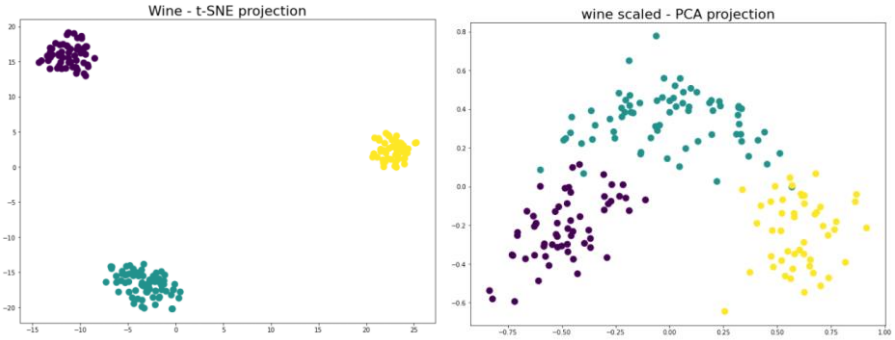


Figura 2: Exemple d'aplicació d'algoritmes de clustering en la base de dades wine dataset.

Font: Projecte de l'assignatura Intel·ligència artificial del GETI dels alumnes María Alcalde, Marc Surís i Guillem Costa

Un altre exemple és l'ús d'*autonecoders.* En aquesta aplicació, les imatges d'entrada es comprimeixen codificant-les i seguidament es descodifiquen. L'objectiu és que les imatges descodificades siguin el més semblant possibles a les reals. Un dels usos dels *autoencoders* és eliminar soroll de les imatges. En la figura 3 es pot veure a la part

superior imatges de dígits amb molt de soroll i a la part inferior les imatges descodificades sense quasi soroll [11].

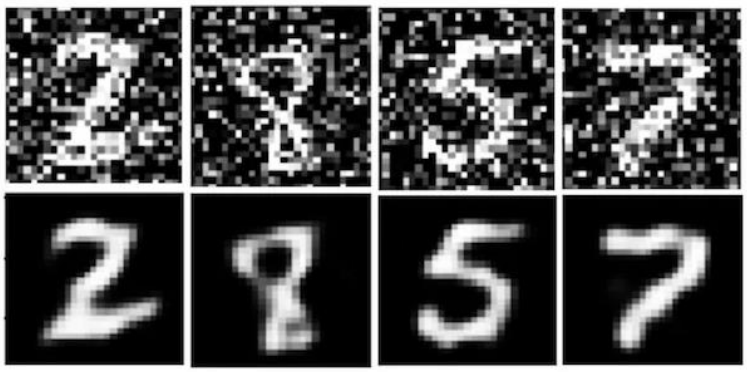


Figura 3: Exemple d'ús d'autoencoders en la base de dades MNIST

Font: Understanding Autoencoders using Tensorflow (Python) [12]

- **Aprenentatge semi-supervisat:** És una barreja entre l'aprenentatge supervisat i el no supervisat. Això és pel fet que una part de les dades d'entrada estan etiquetades i l'altre no. Dintre de l'aprenentatge semi-supervisat existeix una tècnica anomenada *pseudo-labeling*. Consisteix a entrenar el model primer com es faria en un aprenentatge supervisat (amb dades etiquetades) i després utilitzar el model entrenat per predir el resultat de les dades no etiquetades. Amb aquests resultats s'etiqueten aquestes dades. Finalment, es torna a entrenar el model amb totes les dades, les que ja es tenien i les que s'acaben d'etiquetar. Aquest tipus d'aprenentatge és molt útil per bases de dades en què no es poden obtenir totes les dades etiquetades o per aquelles en què l'etiquetatge requereix molts recursos [13].

- **Aprenentatge reforçat:** A diferència dels anteriors, aquest tipus d'aprenentatge es basa en la prova i error i no se li subministra un conjunt de dades, sinó que l'ordinador aprèn a partir de la interacció amb un món virtual o real. Haurà de fer front a una situació similar a un joc en què a partir de prova i error intentarà trobar una solució al problema. Amb l'objectiu de què la màquina realitzi el que vol el programador, aquesta obtindrà una recompensa o una penalització en funció de les seves accions. El seu objectiu és maximitzar les seves recompenses. El programador no li donarà pistes ni informació al model de com resoldre el problema, simplement definirà les regles del joc. Començarà provant amb tècniques senzilles i acabarà fent tècniques sofisticades. Una aplicació de l'aprenentatge reforçat, és la utilització d'aquests models en vehicles autònoms. Com el programador no és capaç de predir tot el que podria passar en una carretera, es fa una simulació en un món virtual en què el model s'entrena amb un sistema de recompenses per aconseguir que el cotxe vagi al punt indicat, seguint les normes de conducció i donant molta importància a la seguretat [14].

3.3 Deep learning

L'aprenentatge profund (en angles *deep learning*), és un subcamp de l'aprenentatge automàtic que utilitza algoritmes inspirats en l'estructura i funció de les xarxes neuronals del cervell.

Les xarxes neuronals que s'utilitzen estan formades per una sèrie de capes cada vegada més complexes que simulen el funcionament bàsic del cervell amb les neurones. D'aquí prové el terme "deep", que en anglès significa profund. Com més capes té la xarxa neuronal més profund és el model. Aquestes xarxes són anomenades xarxes neuronals artificials [15].

3.4 Xarxes Neuronals Artificials

Les xarxes neuronals artificials són sistemes informàtics que es componen d'un conjunt d'unitats connectades anomenades neurones que s'organitzen en el que anomenem capes. Com ja s'ha comentat, estan inspirades en les xarxes neuronals biològiques que constitueixen els cervells dels humans [16].

Cada connexió entre neurones transmet un senyal d'una neurona a l'altra. La neurona receptora processa el senyal i l'envia cap a les que estan connectades a la xarxa. Les neurones també es coneixen habitualment com a nodes.

Cada connexió entre neurones té un pes que s'ajusta a mesura que avança l'aprenentatge. Aquest pes fa augmentar o disminuir la força del senyal en una connexió entre neurones.

Existeixen diferents capes que poden realitzar diverses transformacions a les seves entrades. Els senyals viatgen des de la primera capa (*input layer* o capa d'entrada) fins a la darrera capa (*output layer* o capa de sortida). Entre la capa d'entrada i de sortida normalment hi ha diverses capes anomenades *hidden layers*, que poden ser de diferents tipus [17].

En la figura 4 es pot veure un esquema d'una xarxa neuronal artificial bàsica. Els nodes o neurones es representen amb cercles i cada columna de cercles forma una capa. A l'esquerra hi ha l'*input layer* formada per 3 nodes. En aquestes capes hi ha tants nodes com nombre d'entrades a la xarxa. La capa central és una *hidden layer*, el nombre de neurones d'aquestes capes s'escull arbitràriament depenent de la seva funció. Finalment, a la dreta tenim l'*output layer* amb un node. Cada node d'aquesta capa correspon al nombre de sortides desitjades de la xarxa.

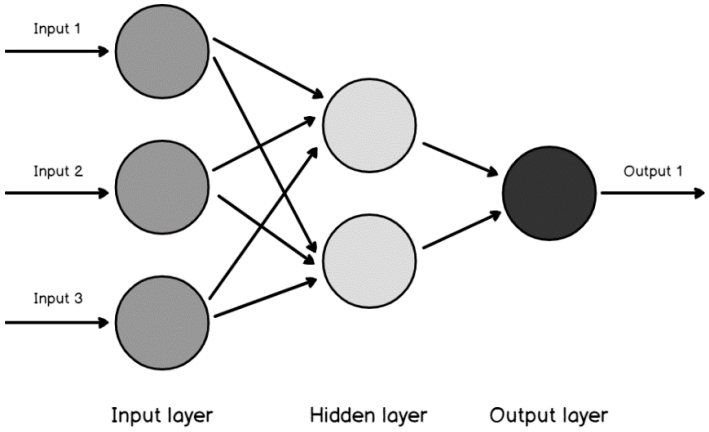


Figura 4: Exemple d'una xarxa neuronal artificial

Font: Implement Artificial Neural Networks (ANNs) in SQL Server [18]

Un possible ús per aquesta xarxa seria un exemple típic de regressió, on s'intenta predir el preu dels habitatges a partir de les seves dimensions, la ubicació i el nombre de veïns. Les entrades serien els atributs, dimensió, ubicació i nombre de veïns (cada un li correspondria un node de l'*input layer*, en total 3) i la sortida seria única, el preu de l'habitatge. Com només hi hauria una sortida només es necessitaria un node a l'*output layer*.

Cada node està connectat amb tots els nodes de la següent capa. Aquesta connexió té associada un valor numèric que s'anomena pes. Quan un la xarxa rep un input a un dels nodes de l'*input layer*, aquest s'envia per les connexions a les neurones de la següent capa. En cada una de les connexions aquest input es multiplica pel pes d'aquesta connexió i arriba al següent node. En aquest es fa el sumatori de tots els números que rep de cada connexió i es calcula la sortida d'aquest node amb una funció d'activació. L'expressió de la funció d'activació és la de l'equació 1.

$$sortida\ node = funció\ d'activació\ (sumatori\ d'entrades + bias) \qquad (1)$$

Equació 1: *Sortida d'un node*

Com es pot veure en l'equació 1, el sumatori d'entrades del node se suma amb un element anomenat *bias*, la utilitat d'aquest terme s'explicarà en l'apartat 3.4.5.

Un cop obtenim la sortida d'un node determinat, aquest és el valor que es passa com a entrada als nodes de la capa següent. Aquest procés continuarà fins a arribar a l'*output layer i* s'anomena *forward propagation*.

3.4.1 Funció d'activació

De funcions d'activació hi ha moltes, però les més utilitzades i les que s'utilitzen en aquest treball són les següents [19]:

- ***Linear*:** També anomenada identitat o no activació, pel fet que es multiplica el sumatori de les entrades per 1. S'utilitza en problemes de regressió on es busquen solucions lineals

- ***ReLu*:** És la funció d'activació més utilitzada. Retorna el màxim del sumatori d'entrades més el *bias* o 0 (equació 2).

$$relu(x) = \max(0, x) \qquad (2)$$

Equació 2: *Funció d'activació ReLu*

Per tant, com es pot veure en la figura 5, si l'entrada és un nombre negatiu, passarà a ser 0, i si és un nombre positiu, es mantindrà igual.

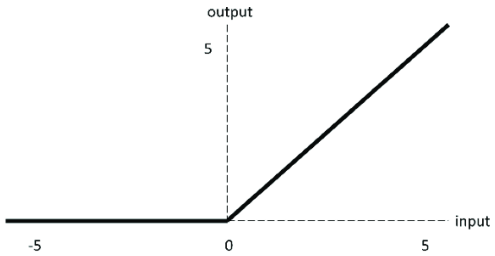


Figura 5: Corba de la funció d'activació ReLu

Font: researchgate.net [20]

La idea és que com més positiva sigui una neurona més activada estarà.

- ***Sigmoid*:** Donada una entrada, si aquesta és molt positiva, la transforma en 1, si pel contrari és molt negativa, la transforma en 0 i si és propera a 0, la transforma en un nombre entre 0 i 1 (equació 3).

$$sigmoid(x) = \frac{e^x}{e^x + 1} \qquad (3)$$

Equació 3: *Funció d'activació sigmoid*

Per tant, com es pot veure en la figura 6, la sortida d'aquesta neurona serà sempre un número entre 0 i 1.

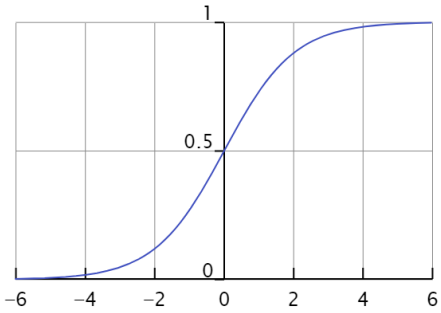


Figura 6: Corba de la funció d'activació sigmoid

Font: Machine Learning & Deep Learning Fundamentals [19]

- ***Softmax:*** S'utilitza per problemes de classificació on hi ha dues o més classes. Aquesta funció calcula les probabilitats relatives. Per exemple, en una xarxa neuronal en la qual donada una imatge d'un animal s'hagi de classificar aquesta en dues classes, 0 si no és un animal i 1 si ho és. Si en l'última capa hi ha una funció d'activació *softmax*, el resultat de les sortides dels dos nodes (hi ha dos nodes perquè hi ha dues classes) serà la probabilitat de què una entrada sigui de la classe 1 o de la classe 0. Ficant el cas que la imatge és d'una zebra i la sortida del node corresponent a la classe 1 és 0,95 i la sortida de l'altre 0,05, voldrà dir que la xarxa neuronal creu que la imatge és amb un 95% de certesa un animal, i 5% de què no ho és. Per tant, la funció *softmax* té en compte les entrades dels altres nodes de la capa i la suma d'outputs de tots els nodes d'aquesta capa ha de donar 1. L'equació és la següent:

$$soxtmax(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} \quad (4)$$

Equació 4: *Funció d'activació softmax*

On x_i és el sumatori d'entrades del node i $\sum_j e^{x_j}$ és el sumatori de l'exponencial de tots els sumatoris d'entrades dels nodes de la capa [21] [22].

Les funcions d'activació estan inspirades en l'activitat del nostre cervell. Quan rebem un estímul, diverses neurones s'activen. Per exemple, si olorem alguna cosa que ens agrada, certes neurones s'activen, en canvi si olorem alguna cosa en mal estat, provocarà que s'activin d'altres. La idea és que com més positiva és la neurona, més activada està.

El principal objectiu de les funcions d'activació és aplicar la no linealitat en el model. Normalment, la major part dels problemes que es ressolen amb xarxes neuronals artificials són no lineals on la solució és complexa. Aquí és on entren les funcions d'activació. La majoria de les funcions d'activació no són lineals i s'escullen d'aquesta manera a propòsit.

3.4.2 Entrenament

Com ja s'ha comentat anteriorment, les xarxes neuronals necessiten un procés d'entrenament perquè el model aprengui a resoldre el problema. Bàsicament durant aquest procés s'està intentant resoldre un problema d'optimització dels pesos. L'entrenament estarà format per diversos *epoch*, que són el nombre de vegades que el model veu tota la base de dades. En cada *epoch* el model anirà variant els pesos aproximant-se als òptims.

Els pesos s'inicialitzen amb valors aleatoris i s'optimitzen mitjançant un algoritme d'optimització. Hi ha diversos algoritmes, cadascun variarà els pesos en funció del seu objectiu. El més utilitzat i el que s'utilitzarà en aquest treball és el SGD (algorisme del gradient estocàstic).

L'objectiu del SGD és minimitzar el que s'anomena funció de pèrdues. Per tant, anirà variant els pesos del model fins que aquesta funció s'aproximi al mínim.

Existeixen diverses funcions de pèrdues i dependrà d'aquesta funció i de l'algoritme d'optimització de com variïn els pesos.

Aquestes funcions calculen un error. Aquest és la diferència entre el valor que s'obté de l'*output layer* i el valor real (equació 5). Tornem a l'exemple de la xarxa neuronal que classificava imatges com a classe 1 si era un animal o 0 si no ho era. Recordem que per una imatge d'una zebra havia retornat que era un animal amb un 95% de seguretat. Per tant, com la imatge sí que és d'un animal i la imatge està etiquetada com a classe 1, l'error en aquest cas és $1 - 0{,}95 = 0{,}05$. L'altre node, en canvi havia retornat que no era un animal amb un 5% de seguretat. L'error en aquest cas és $1 - 0{,}05 = 0{,}95$.

$$|error| = output - \ etiqueta \qquad (5)$$

Equació 5: *Error de l'output*

Al final de cada *epoch*, quan totes les dades de la base de dades hagin passat pel model, l'error o pèrdua es calcularà per cada entrada individual amb el valor obtingut de l'output i el valor de l'etiqueta. En aquest moment, es calcularan els pesos en funció de la funció d'optimització. No sempre es calculen de nou els pesos al final de cada *epoch*. Es pot definir que el model torni a calcular-los més sovint, cada vegada que el model veu un nombre determinat de dades. Aquest valor és l'anomenat *batch size*, i indica el nombre de dades que passen per la xarxa a la vegada. Com més gran sigui, més ràpid serà el procés d'entrenament, però pitjor serà la qualitat dels resultats. Dependrà de les especificacions de l'ordinador [23].

3.4.3 Funció de pèrdues

La funció de pèrdues més utilitzada és la MSE o error quadràtic mig [24].

Consisteix a calcular l'error al quadrat per cada input de la xarxa neuronal artificial (equació 6).

$$MSE(input) = (output - \ etiqueta) \cdot (output - \ etiqueta) = error^2 \qquad (6)$$

Equació 6: *Fórmula MSE*

A continuació es fa la mitjana aritmètica amb el resultat de cada input (equació 7).

$$MSE = \frac{1}{n} \cdot \sum_{i=1}^{n} MSE(input_i) \qquad (7)$$

Equació 7: *Mitjana aritmètica del resultat de l'equació 5 per cada input*

On n és el nombre d'inputs de la xarxa neuronal o dades de la base de dades.

El valor obtingut de l'equació 7, serà el que SGD intentarà minimitzar i anirà canviant en cada *epoch*, ja que els pesos variaran.

En el model que s'ha utilitzat en aquest treball no s'ha emprat aquesta funció. S'han fet ús d'altres. Una d'elles és la SmoothL1_loss [25].

3.4.4 SGD

Com ja s'ha comentat anteriorment, la manera en la qual la xarxa neuronal aprèn és mitjançant l'actualització dels pesos al final de cada *epoch*. Aquest càlcul el fa mitjançant una tècnica anomenada *backpropagation* (en català, propagació inversa). Una vegada el model calcula les pèrdues amb la funció de pèrdues, calcula la derivada d'aquesta funció respecte cada un dels pesos. Aquest valor indicarà la direcció en la qual s'ha de moure el pes per minimitzar les pèrdues [26]. En la figura 7 es pot veure un esquema d'aquest procés.

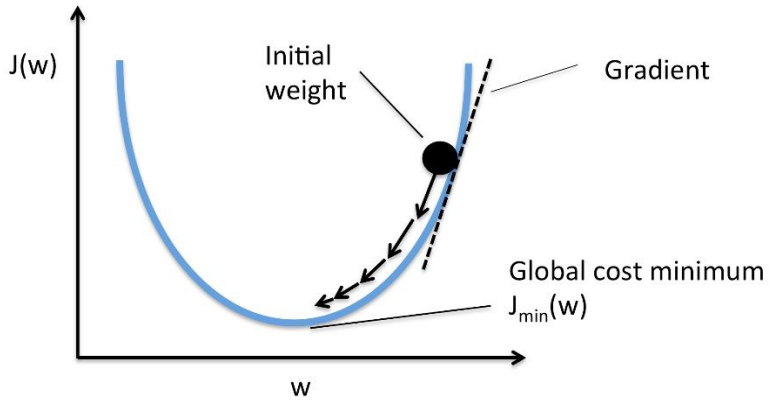


Figura 7: Procés realitzat pel SGD

Font: Gradient Descent and Stochastic Gradient Descent [27]

A continuació es multiplica el valor de la derivada per un valor anomenat *learning rate*, que sol ser un número entre 0,01 i 0,0001. Aquest valor indica quant variarà el valor del pes per assolir el mínim d'error. Aquest valor no ha de ser molt gran per no superar el mínim, ni tampoc molt petit, sinó que no s'arribaria mai. En la figura 8 es pot veure un exemple de com en el model de l'esquerra es necessitarà moltes *epoch* per assolir el mínim i en canvi el de la dreta està constantment superant aquest valor, per tant mai arribarà a ell. El del mig és un exemple de model amb un bon *learning rate*.

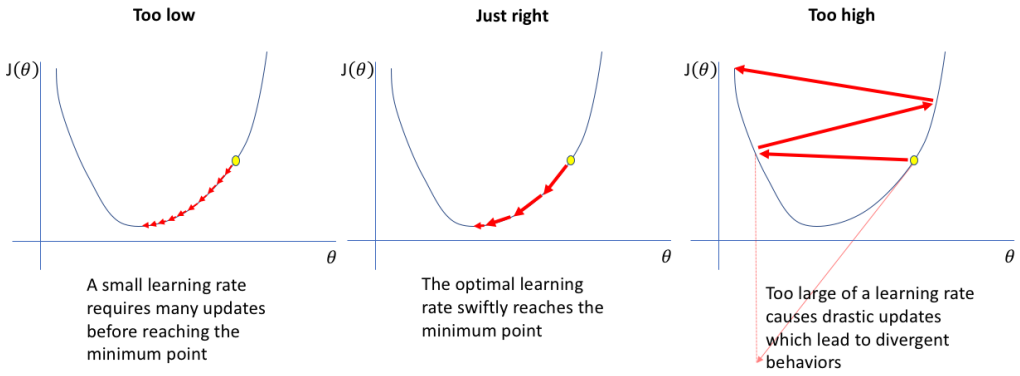


Figura 8: Procés realitzat pel SGD en 3 models amb diferent learning rates

Font: Setting the learning rate of your neural network [28]

Finalment, per obtenir el nou pes es fa la diferència entre el valor del pes actual amb el valor de la multiplicació de la derivada amb el *learning rate (equació 8).*

$$Nou\ pes = pes\ antic - (learning\ rate \cdot derivada) \quad (8)$$

Equació 8: *Fórmula pes*

Aquest procés es durà a terme per cada pes de cada connexió entre nodes al final de cada *epoch*. Començant per les capes de més a la dreta i acabant per la de més a l'esquerra. Per això aquest procés s'anomena *backpropagation* o retropropagació.

3.4.5 Bias

Com abans s'ha mencionat, el bias [29] és un terme que apareix en el càlcul de la sortida d'un node (vegeu l'equació 1).

Cada node té associat un *bias*. Aquest és un terme que de la mateixa manera que els pesos, la xarxa neuronal torna a calcular al final de cada *epoch*. El *bias* determina si una neurona s'activarà o no, o en quina mesura. Ens fa saber quan s'activa una neurona de manera significativa. L'addició del *bias* acaba augmentant la flexibilitat d'un model per ajustar-se a les dades donades. A continuació s'explicarà el funcionament del *bias* amb un exemple:

Suposem que es té una xarxa neuronal amb una *input layer* amb 2 nodes, i a continuació hi ha una *hidden layer* d'1 node. El primer node de l'*input layer* té un valor de 2 i el segon de 3 i els pesos de les connexions d'aquests nodes amb el node de la *hidden layer* són de -0,5 i 0,3 respectivament. En la figura 9 es pot veure una representació d'aquesta xarxa.

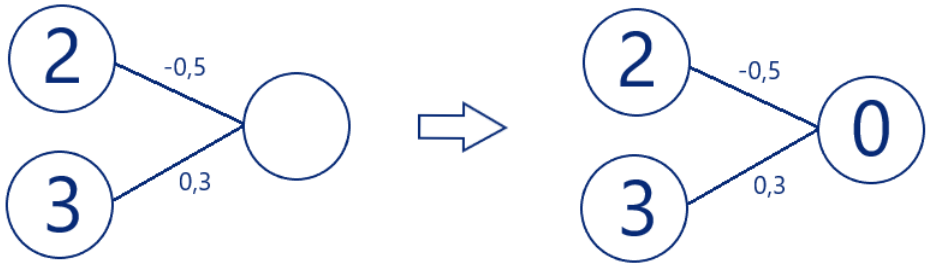


Figura 9: Xarxa neuronal artificial

Font: Realitzada pel mateix alumne

El sumatori d'entrades al node de la *hidden layer* serà:

$$2 \cdot (-0{,}5) + 3 \cdot (0{,}3) = -0{,}1 \qquad (9)$$

Equació 9: *Sumatori d'entrades al node de la hidden layer*

Si el *bias* fos 0, l'output d'aquest node es calcularia amb la funció d'activació:

$$sortida\ node = funció\ activació(-0{,}1) \qquad (10)$$

Equació 10: *Output d'un node*

Si per exemple la funció d'activació fos la *ReLu,* la sortida seria igual a:

$$sortida\ node = relu(-0{,}1) = \max(0, -0{,}1) = 0 \quad (11)$$

Equació 11: *Output d'un node amb funció d'activació ReLu*

Quan en un node, la sortida pren valor de 0, es considera que la neurona no està activada, per tant cap informació que rebi aquesta neurona es transmetrà a la xarxa.

El *bias* el que fa és sumar un valor al sumatori d'entrades del node per activar aquest node en el cas que el resultat de la funció d'activació sigui 0 o inferior. En aquest cas, per exemple, si es volgués activar la neurona, s'establiria un *bias*>0,1.

Però com ja s'ha comentat, el *bias* és un valor que igual que els pesos es calcula en cada *epoch*, que s'inicialitza aleatòriament i que per tant, no l'elegeix el programador. En definitiva, permetrà al nostre model aprendre quan activar i quan no activar cada neurona. Això el farà ser més flexible.

3.5 Xarxes neuronals convolucionals (CNN)

La xarxa neuronal artificial que s'ha utilitzat en aquest projecte és una xarxa neuronal convolucional. El seu objectiu és de detectar els caps dels pops blancs.

Una xarxa neuronal convolucional és un tipus de xarxa neuronal artificial utilitzada per analitzar imatges i poder detectar patrons. Aquesta detecció de patrons és el que fa que les xarxes CNN siguin tan útils per a l'anàlisi d'imatges [30].

El que fa que aquest tipus de xarxa pugui detectar patrons és que estan formades per unes *hidden layers* anomenades capes convolucionals.

A continuació s'explicarà el funcionament de les diferents capes que es troben en una CNN:

Capa convolucional

La seva funció és extreure les característiques més importants d'una imatge i reduir-la perquè sigui més fàcil processar-la. Recordant allò comentat, són capaces de detectar patrons en les imatges, això ho fa gràcies als filtres. Per cada capa convolucional s'ha d'especificar el nombre de filtres. A diferència de les capes explicades en l'apartat de les xarxes neuronals artificials, aquestes no estan formades per nodes, sinó que els nodes se substitueixen per filtres.

Els patrons que detecta poden ser des d'arestes, cercles, corbes, formes, colors fins a objectes específics com cares de gossos, ulls, ocells, etc. Com més profunda sigui la xarxa, més sofisticats es tornen els filtres.

Els filtres són matrius amb unes dimensions concretes (normalment petites, per exemple 3 x 3) que s'inicialitzen amb valors aleatoris. Aquests valors es calculen de nou en cada *epoch*. Tenen una funció similar als pesos de les xarxes neuronals artificials que s'han explicat en l'apartat 3.4.

Com qualsevol altre capa, rep un input (una imatge, que en definitiva és una matriu dels valors dels píxels), la transforma i retorna un output que s'envia a la següent capa. La transformació que succeeix s'anomena operació de convolució. Aquesta operació consisteix a subdividir els píxels de la imatge d'entrada en totes les matrius possibles amb la dimensió del filtre i a continuació fer el producte intern de cada matriu amb el filtre (equació 12). El resultat serà una matriu formada pels valors dels píxels d'una nova imatge que s'anomenarà mapa de característiques.

El producte intern de A i B consisteix a multiplicar cada element de la matriu A amb el que està situat en la mateixa posició en la matriu B i sumar els resultats de les multiplicacions.

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

$$B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

$$Producte\ intern\ A\ i\ B = a_{11} \cdot b_{11} + a_{12} \cdot b_{12} + a_{21} \cdot b_{21} + a_{22} \cdot b_{22} \qquad (12)$$

Equació 12: *Producte intern de dues matrius*

En la figura 10 es pot veure un exemple d'aquesta operació de convolució en una imatge de 6 x 6 x 3 píxels. El 3 fa referència al fet que en aquest cas, la imatge és en clor RGB. En aquests casos la imatge està formada per tres matrius. El filtre és una matriu 3 x 3. El procediment és el mateix, cada matriu fa el producte intern de totes les submatrius amb dimensions 3 x 3 amb el filtre (o *kernel* en angles) però en el cas d'imatges RGB se suma el resultat de les tres matrius. Com es pot veure en la imatge, les capes convolucionals també tenen *bias*, i igual que els valors dels filtres es calcularan de nou en cada *epoch* perquè la xarxa aprengui.

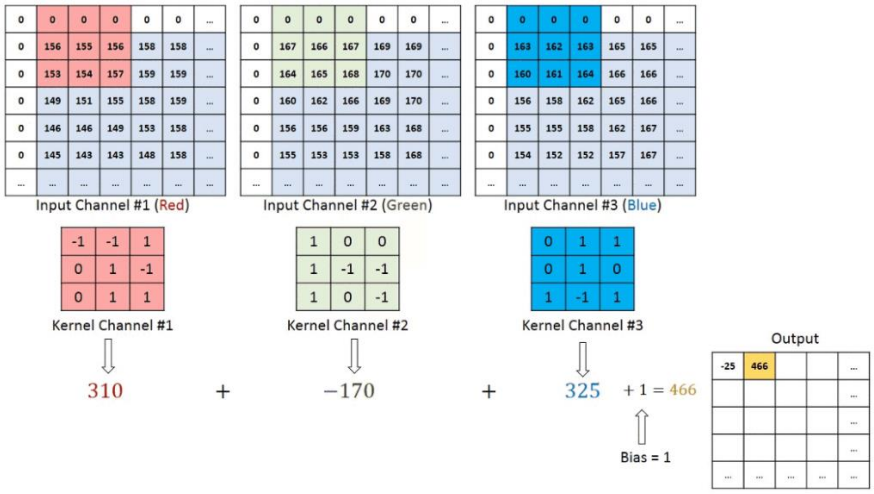


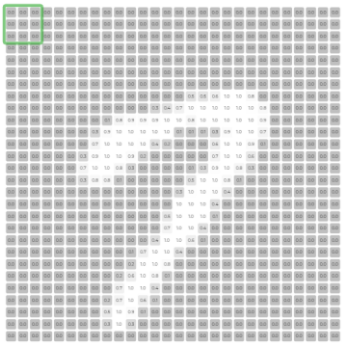
Figura 10: Exemple d'operació convolucional d'una imatge 6 x 6 x 3 amb un filtre 3 x 3

Font: A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way [31]

Com es pot veure en la figura 10, la matriu resultant de l'operació convolucional no és de la mateixa dimensió que la matriu original. Això pot ser un problema per imatges en què hi ha informació rellevant en les vores. A més, en cada capa la imatge es farà més i més petita. Aquest problema se soluciona amb una tècnica anomenada *zero padding* [32]. Aquesta tècnica consisteix a afegir una vora de píxels tots amb valor zero al costat de les vores de les imatges d'entrada. Això farà que la matriu resultant sigui de la mateixa dimissió que la de l'entrada i no es perdi informació rellevant en les vores de la imatge.

En la figura 11 hi ha un conjunt d'imatges de base de dades MNIST on s'ha fet el producte convolucional amb diferents filtres:

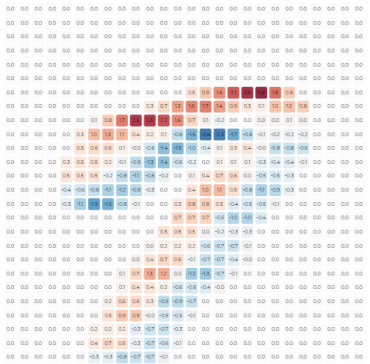
La imatge original és el número 7:



El resultat de fer l'operació convolucional amb els filtres és el següent:

1. Filtre vora superior

$$\begin{pmatrix} -1 & -1 & -1 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$



2. Filtre vora esquerra

$$\begin{pmatrix} -1 & 1 & 0 \\ -1 & 1 & 0 \\ -1 & 1 & 0 \end{pmatrix}$$

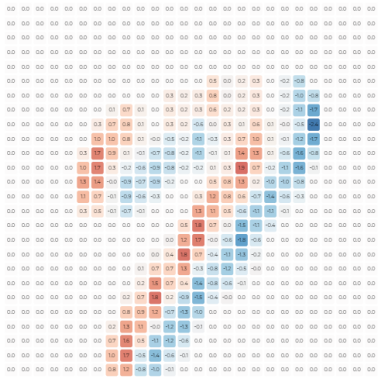


Figura 11: Exemple de productes convolucionals

Font: DEEPLIZARD Demo [33]

Com es pot veure en la figura 11, el filtre el que fa és ressaltar les vores de les imatges. Aquests filtres són molt senzills, però com ja s'ha comentat com més profunda sigui la capa de la CNN, més complexos seran els filtres.

Capa d'agrupació (*pooling*)

En molts casos la quantitat de matrius que una CNN ha de processar a la vegada és molt gran. Les capes d'agrupació s'ocupen de disminuir la potència computacional requerida per processar les dades a través de la reducció de dimensionalitat, conservant la informació més rellevant.

Hi ha de dos tipus, la *max pooling* i la *average pooling*, la primera és la més utilitzada i la que millors resultats dona [31].

Per definir una capa d'agrupació s'han de determinar els següents paràmetres:

- Dimensió n x n del filtre
- *Srtide*: Nombre de píxels que es vol que es mogui el filtre mentre llisca per la imatge.

Per explicar el procediment, suposem una matriu 4 x 4, un filtre 2 x 2 i un *stride* de 2. El filtre és de tipus *max pooling*. L'operació és la que es pot veure en la figura 12.

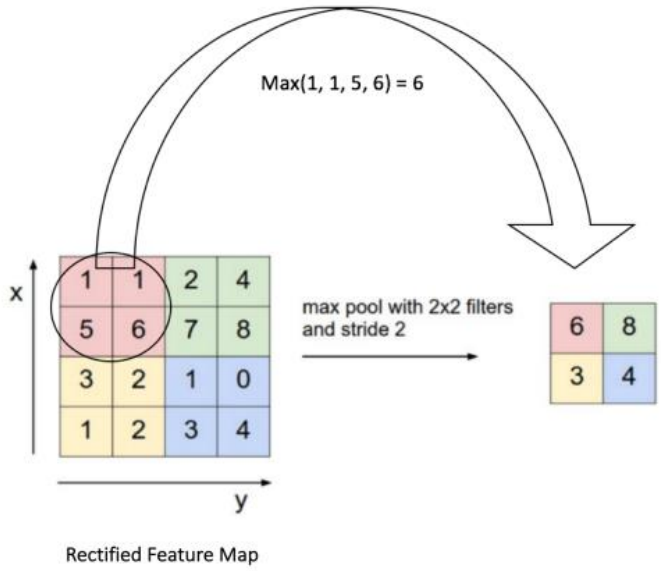


Figura 12: Resultat d'una capa max pooling amb filtre 2 x 2 i stride de 2

Font: An Intuitive Explanation of Convolutional Neural Networks [34]

Per obtenir la matriu de sortida, s'agafarà la primera matriu 2 x 2 de la imatge original (matriu blava) i s'escull el màxim = 6. A continuació es mou 2 píxels. Ara la matriu que s'agafarà serà la groga. S'agafa el màxim = 8. Com ja no pot anar més a la dreta baixa 2 píxels situant-se a la matriu de color

salmó i torna a agafar el màxim = 4. Finalment es mou dos píxels a l'esquerra i agafa el màxim = 3. L'operació acaba aquí perquè no es pot moure 2 píxels a la dreta ni pot baixar.

Les capes *average pooling* funcionen igual, però en comptes d'agafar el màxim de cada submatiru s'agafa la mitjana aritmètica.

Altres motius d'utilitzar capes d'agrupació és que eliminen el soroll en les imatges i redueixen l'*overfiting* (el significat d'aquest terme s'explica en l'apartat 5.1).

Capa d'activació (*ReLu*)

S'encarrega d'introduir la no linealitat a la CNN, ja que totes les operacions que s'han explicat fins ara han estat sumes i multiplicacions. El seu funcionament és idèntic al de la funció d'activació ReLu explicat anteriorment. Canvia tots els píxels del filtre amb valor més petit que zero per 0. Aquestes capes se situen generalment darrere d'una capa convolucional.

Capa de deserció (*dropout*)

Desactiva aleatòriament algunes neurones posant a zero una fracció de les seves entrades. Per tant, la xarxa es veu obligada a classificar un input encara que algunes activacions es desactivin. Això fa que la CNN sigui més robusta i menys propensa a tenir *overfitting*.

Capa FC

El seu funcionament és el mateix que s'ha explicat en l'apartat 3.4 per les *hidden layers*. Està formada per un conjunt de neurones. Es diu FC o *fully-connected* perquè cada neurona està connectada amb totes les neurones de la següent capa i l'anterior. Aquestes connexions tenen un pes. Cada neurona té una funció d'activació. En problemes de detecció d'objectes, se solen ficar al final perquè permeten recollir tota la informació obtinguda durant les capes anteriors i executar els algoritmes de *forward-propagation* i *backward-propagation*.

Capa s*oftmax*

L'última capa d'una CNN de classificació sol ser una capa FC amb la funció d'activació s*oftmax* (explicada en l'apartat 3.4.1). Determinen la probabilitat de què existeixi un determinat objecte en la imatge.

Capa flatten

Transforma una entrada 2D en 1D. És necessària per transformar les sortides de les capes convolucionals (2D) en entrades per les capes FC (1D).

En la figura 13 es pot veure un exemple d'una CNN que classifica imatges de vehicles. Com es pot observar, les primeres capes són un conjunt de capes convolucionals, capes *ReLu* i capes p*ooling*. Seguidament, hi ha una capa *flatten* per passar les sortides d'aquesta a valors d'una dimensió, una capa FC i per últim una capa softmax que permet classificar les imatges.

Les capes de deserció poden estar situades darrere de qualsevol capa convolucional, *pooling* o FC.

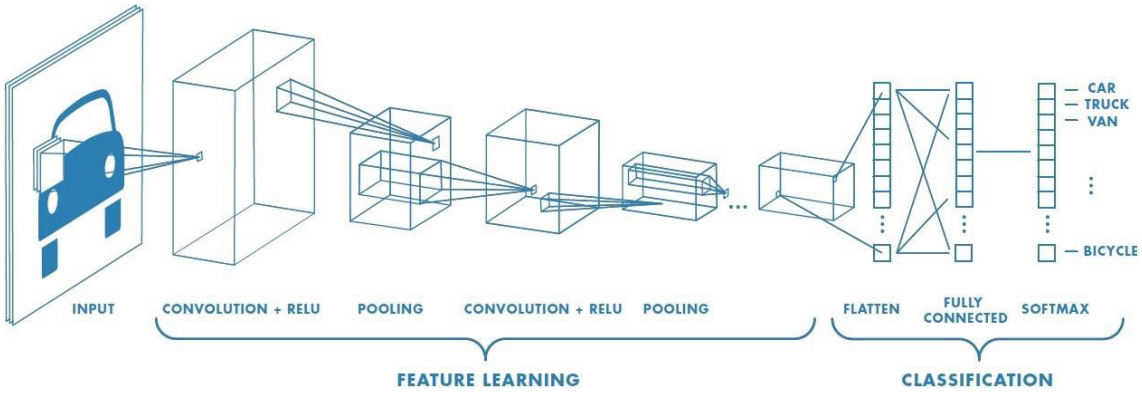


Figura 13: Exemple CNN

Font: A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way [31]

3.6 Mask R-CNN

Mask R-CNN és una xarxa neuronal convolucional utilitzada per la segmentació d'instàncies d'objectes i és la CNN que s'ha utilitzat en aquest treball. Detecta eficaçment objectes d'una imatge alhora que genera una màscara de segmentació d'alta qualitat per a cada instància.

El que es vol aconseguir en aquest treball és que donada una imatge d'una caixa de pops blancs, aquesta sigui retornada amb les màscares dels caps dels pops blancs.

El terme màscara (*mask* en anglès) fa referència al fet que encercla els objectes que detecta. A més, per cada objecte que detecta crea una *bounding box* (caixa de delimitació en català) i retorna la probabilitat de què la predicció que hagi fet sigui certa. En la figura 14 es pot veure un exemple d'allò que s'obté. Cada persona i avió té una màscara per sobre, una *bounding box* al seu voltant, el nom de la classe a la qual pertany i la probabilitat de què realment pertanyi a aquesta classe.

Figura 14: Output Mask R-CNN

Font: Mask R-CNN for Object Detection and Segmentation [1]

Per tant, per cada imatge que entri a la CNN, aquesta haurà de retornar:

- Una llista de *bounding boxes*.
- Una etiqueta assignada a cada *bounding box* indicant la classe de l'objecte.
- Una probabilitat per cada etiqueta i *bounding box*.
- Una llista de màscares.

Com en les xarxes neuronals artificials explicades en l'apartat 3.4, on cada entrada a la base de dades s'ha d'etiquetar (per exemple, en tasques de classificació s'ha d'indicar la classe a la qual pertany cada input). Per utilitzar la Mask R-CNN, cada imatge s'ha d'etiquetar indicant per cada objecte que es vol detectar, la classe a la qual correspon i la seva màscara. La Mask R-CNN només detectarà els objectes corresponents a les classes definides.

A continuació s'explicarà el seu funcionament [35]:

1. Utilitzar una CNN ja entrenada per a classificació amb una base de dades molt gran, com pot ser ImageNet i utilitzar la sortida d'una de les capes convolucionals. Una de les xarxes neuronals convolucionals més utilitzades és ResNet. Per tant, s'obtindrà per cada imatge un mapa de característiques (output de la capa convolucional) d'una dimensió menor a la imatge original, però més profund (serà tan profund com nombre de filtres tingui l'última capa convolucional). En aquest mapa de característiques, s'haurà codificat tota la informació de la imatge mantenint la ubicació de les "coses" que ha codificat en relació amb la imatge original. Per exemple, si hi hagués un pop blanc a la part superior esquerra de la imatge, les capes convolucionals s'haurien activat i la informació d'aquest pop blanc continuaria estant a la part superior esquerra del mapa de característiques.

2. A continuació, hi ha el que s'anomena Xarxa de Propostes de Regió (RPN).
   A partir del mapa de característiques de cada imatge, es genera en cada píxel un conjunt d'*anchors* que tenen com a centre aquest píxel.
   Els *anchors* són *bounding boxes* de dimensions fixes que es col·loquen al llarg de la imatge amb diferents mides i relacions d'alçada i amplada que s'utilitzaran com a referència quan s'intenta predir per primera vegada les ubicacions dels objectes.
   Encara que els *anchors* es generen en el mapa de característiques de cada imatge, les seves dimensions fan referència a les de la imatge original.
   En la figura 15 es pot veure els centres dels *anchors* en una imatge

Figura 15: Centres dels anchors en una imatge

Font: Faster R-CNN: Down the rabbit hole of modern object detection [35]

El que la CNN intentarà predir no seran les dimensions de les *bounding boxes*, sinó el $\Delta_{x\ centre}$, $\Delta_{y\ centre}$, Δ_{altura} i Δ_{gruix} respecte a les dimensions d'un *anchor*. Cada *anchor* es definirà com x_{centre}, y_{centre}, altura i gruix.

Un cop s'han creat tots els *anchors*, RPN, selecciona els que realment contenen un objecte. Això ho fa retornant per cada *anchor* la probabilitat de què contingui un objecte (no té importància la classe de l'objecte) i una regressió de la *bounding box* per ajustar millor els *anchors* a l'objecte que prediu. La manera de fer-se és utilitzant el mapa de característiques de cada imatge. Primer amb una capa convolucional amb 512 filtres 3 x 3 i després amb dues capes en paral·lel, una de classificació i l'altre de regressió amb tants filtres 1 x 1 com *anchors* tingui la imatge.
La de classificació retorna la probabilitat de què cada *anchor* contingui un objecte i la de què no.
La de regressió retorna quatre prediccions: $\Delta_{x\ centre}$, $\Delta_{y\ centre}$, Δ_{altura}, i Δ_{gruix}.
Durant el procés d'entrenament per calcular les pèrdues i fer que la CNN aprengui, se separen els *anchors* en dos grups: Aquells amb un IoU > 0,5 són considerats *foreground* (primer pla), i els que tenen un IoU < 0,1, són considerats *background* (fons).

IoU és l'abreviatura d'*Intersetion over object* (en català, intersecció sobre objecte). Indica quant de precís és l'*anchor* en relació amb la ubicació de l'objecte de la imatge original (es compara la dimensió i la ubicació de l'*anchor* amb la bounding box real).
Per calcular les pèrdues de classificació s'utilitza la classificació que es fa amb el IoU i una tècnica anomenada entropia creuada binària. I per mesurar les de regressió s'utilitzen només els *anchors* classificats com a *foreground* i les *bounding boxes* reals i es calcula l'increment necessari per transformar l'*anchor* en la *bounding box* real. La funció de pèrdues utilitzada és l'anomenada *Smooth L1*, que calcula l'error fent la diferència entre el valor real i el predit. Quan l'error és prou petit, definit per una σ, l'error es considera gairebé correcte i la pèrdua disminueix a un ritme més ràpid.

La gran quantitat d'*anchor*s creats fa que molts se sobreposin sobre el mateix objecte. Per eliminar les propostes duplicades i evitar que es generi més d'una *bounding box* sobre un objecte s'utilitza un algoritme anomenat N*on-Maximum Suppression* (NMS). NMS, ordena tots els *anchor*s sobre un objecte en funció del seu IoU. I suprimeix els *anchors* amb un IoU menor a un valor. Un valor normalment utilitzat és 0,6. En la figura 16 es pot veure un exemple d'abans i després d'aplicar NMS en una imatge.

Figura 16: Imatge abans i després d'aplicar NMS

Font: Object Detection for Dummies Part 3: R-CNN Family [36]

Després d'aplicar NMS, es fa una selecció de tots els *anchors* restants i s'ordenen en funció del IoU. Seguidament, es fa una selecció i es descarta la resta. Nomenament se seleccionen 2000 *anchors*.

3. El mapa de característiques obtingut en el pas 1, s'utilitza com a input per una capa anomenada *RolAlign*. La sortida d'aquesta capa seran tants mapes de característiques com propostes de *bounding boxes* (*anchors* obtinguts al final del pas 2). Consisteix a retallar la part del mapa de característiques que correspon a cada proposta i posteriorment ajustar la seva dimensió, que serà la mateixa per tots els mapes de característiques. La part que s'ha de retallar se selecciona mitjançant la interpolació bilineal, que li permet ser molt precisa i no perdre informació. Per últim es redueix la seva dimensió amb una capa *max pooling*. La

sortida serà de dimensió 7 x 7 x 512 (el 512 prové de què en el pas 2 s'utilitza una capa convolucional de 512 filtres). En la figura 17, es pot veure el procediment.

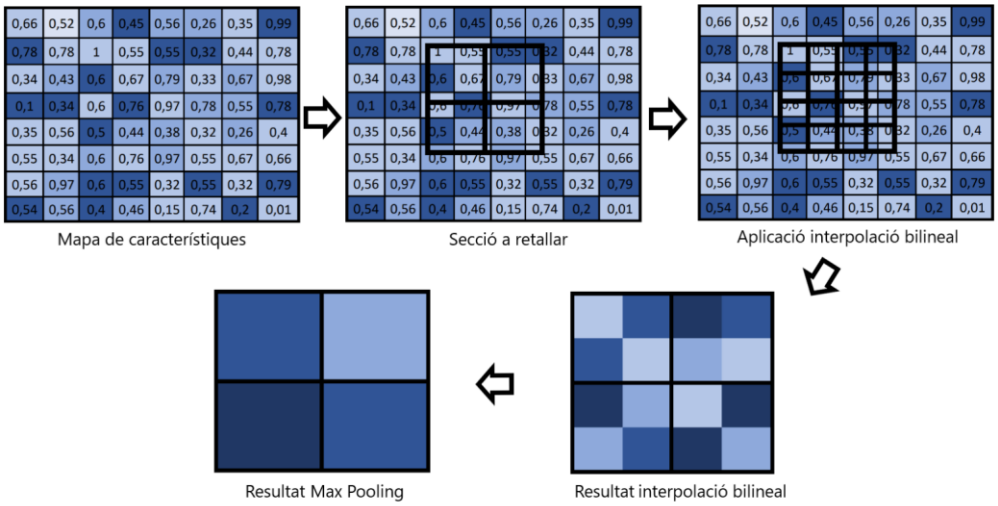


Figura 17: Funcionament del pas 3 per una proposta de bounding box

Font: Realitzada per l'alumne

4. Utilitzar els mapes de característiques obtinguts en el pas 3 per determinar la classe a què pertanyen, determinar i ajustar les *bounding boxes* en funció de la classe predita i determinar les màscares per cada *bounding box*.
   La Mask R-CNN agafa com a inputs els mapes de característiques obtinguts en el pas 3 (un per cada proposta de *bounding box*) i els fa passar per una capa *flatten* per convertir els mapes de característiques de dimensió 7 x 7 x 512 a un vector de 1D. A continuació, utilitza dues capes FC amb 4096 neurones amb la funció d'activació *ReLu*. Per últim, s'utilitza la sortida d'aquesta capa com a entrada per tres capes en paral·lel. Aquestes són les següents:

   - Capa FC amb funció d'activació *softmax* per classificació, amb N+1 neurones. N és el nombre de classes. En aquest treball, només hi haurà una classe i es definirà com a Pop_blanc. Per cada proposta de *bounding box*, aquesta capa retorna la probabilitat de què pertanyi a cada una de les classes. Assigna a cada proposta a la classe amb més probabilitat. La neurona extra correspon a la classe *background* o fons. Les *bounding boxes* amb major probabilitat de ser d'aquesta classe es descartaran.
   - Capa FC utilitzada per regressió amb $4 \cdot N$ neurones. Es vol una sortida corresponent a $\Delta_{x\ centre}$, $\Delta_{y\ centre}$, Δ_{altura}, i Δ_{gruix} per cada classe. Retornarà per cada proposta de *bounding box* el $\Delta_{x\ centre}$, $\Delta_{y\ centre}$, Δ_{altura}, i Δ_{gruix} només de les neurones corresponents a la classe que s'ha assignat en la capa *softmax*. Si la capa *softmax* determina que no correspon a cap classe, no retorna res.

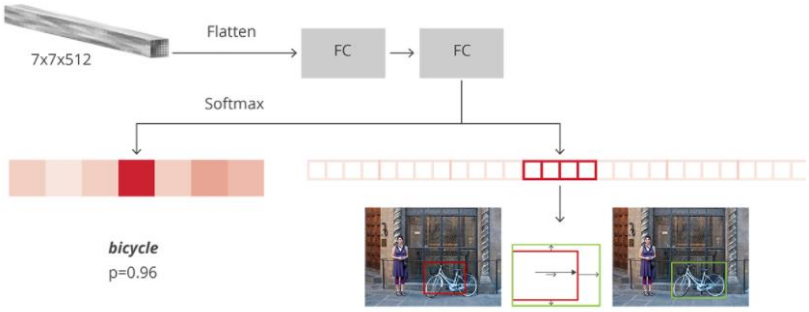


Figura 18: Arquitectura R-CNN

Font: Faster R-CNN: Down the rabbit hole of modern object detection [35]

- Dues capes convolucionals, la primera amb filtres 3 x 3 i funció d'activació *ReLu* i la segona amb filtres 1 x 1. Les entrades són els mapes de característiques de les *bounding boxes*. Aquestes capes classifiquen els píxels de la *bounding box* de la imatge original en funció si pertanyen a l'objecte o formen part del fons de la imatge. Per cada *bounding box*, retorna tantes màscares com classes pel fet que el model intenta aprendre una màscara per a cada classe [37].

En la figura 18 es pot veure l'arquitectura de les dues capes FC.

El càlcul de les pèrdues és molt similar al del pas 2. La principal diferència és que en les pèrdues de regressió s'ha de tenir en compte que pot haver-hi més d'una classe.

Per evitar *bounding boxes* repetides s'aplica NMS en funció de la classe. Aquesta vegada s'agrupen les *bounding boxes* per classes i s'ordenen de més a menys probabilitat de ser de la classe assignada. Per últim, es pot determinar una probabilitat mínima que han de complir les *bounding boxes* per ser considerades d'aquella classe o un límit de *bounding boxes* assignades a cada classe.

Encara que la Mask R-CNN retorna una màscara per cada classe i *bounding box*, finalment només retornarà aquelles màscares de les *bounding boxes* que se seleccionen i les màscares corresponents a la classe que s'ha assignat a la *bounding box*. Per calcular les pèrdues de les màscares es compara la màscara real amb què retorna la Mask R-CNN i s'aplica una funció de pèrdues. La funció de pèrdues utilitzada és l'entropia creuada mitjana binària [38].

Com s'ha comentat en l'apartat 3.4.4, la Mask R-CNN utilitza SGD com a funció d'activació.

La xarxa neuronal convolucional que s'utilitza en el pas 1 és la ResNet101 backbone i empra uns pesos ja entrenats de la base de dades de COCO.

COCO és una base de dades d'imatges especialitzada en detecció i segmentació d'objectes. Està formada per més de 200.000 imatges etiquetades amb 80 classes diferents. L'ús d'aquests pesos farà el procés d'entrenament sigui molt més curt [39].

En la figura 19 es pot veure un esquema del funcionament de la Mask R-CNN.

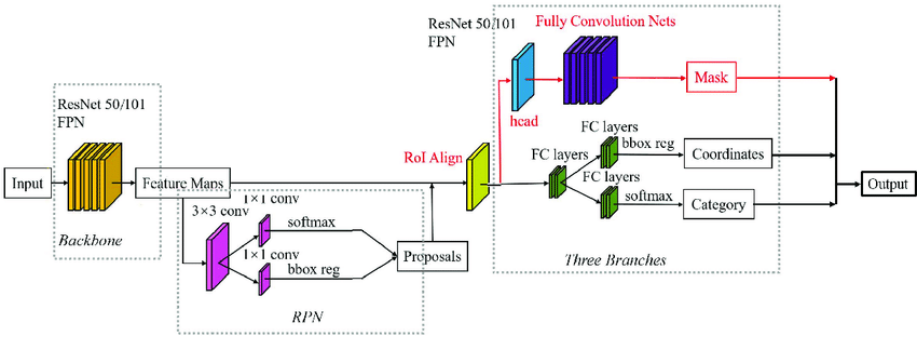


Figura 19: Funcionament Mask R-CNN

Font: researchgate.net [40]

3.7 Tensorflow i Keras

Per fer ús de la Mask R-CNN i realitzar tot el procés d'inferència, en aquest treball s'ha utilitzat el llenguatge de programació python 3. A més, s'ha fet ús de les llibreries TensorFlow i Keras en les versions 1.15.2 i 2.3.1 respectivament.

Aquestes llibreries permeten crear xarxes neuronals artificials amb les diferents capes que s'han explicat en aquest treball, com també visualitzar les pèrdues, fer la inferència, etc.

A continuació es farà una breu explicació d'aquestes dues llibreries:

- **TensorFlow*:*** és una biblioteca de software de codi obert des de 2015, pel càlcul numèric mitjançant gràfics de flux de dades. Va ser desenvolupat originalment per un equip de Google (*Google Brain Team*) per la recerca de *machine learning* i *deep learning*. Actualment TensorFlow és tant general que es pot aplicar a molts altres camps. L'arquitectura flexible de TensorFlow permet utilitzar diferents sistemes d'entrenament i inferència de diferents mides en una àmplia varietat de plataformes de maquinari diferents, com ara màquines individuals que contenen una o més targetes GPU, Android i iOS.

  Els programes en TensorFlow estan basats en grafs de fluxos de dades. Per la creació dels grafs utilitza els tensors. En TensorFlow, un tenor es representa com una matriu unidimensional de números. Les dimensions del tenor es defineixen a partir del rang i la forma. El rang és el nombre de dimensions del vector i la forma és la grandària de cada dimensió [41].

  Els tensors admeten una gran varietat de tipus d'elements, com enters que varien en grandària des de 8 a 64 bits, *floats*, nombres complexos i *strings*.

- **Keras:** És una API (interfície de programació d'aplicacions) de *deep learning* escrita en python. De la mateixa manera que TensorFlow, és de codi obert. Keras es pot executar damunt de TensorFlow, Microsoft Cognitive Toolkit (CNTK) o Theano. Es va desenvolupar per l'experimentació amb xarxes neuronals artificials i va ser dissenyat per l'enginyer François Chollet de Google en 2015 [42].

4 Creació de la base de dades i entrenament del sistema

4.1 Obtenció de les imatges

Les imatges utilitzades durant el projecte corresponen a les caixes de pops blancs (similars a les de la figura 20) que s'han subhastat durant els mesos de febrer i març a les llotges de Llançà i Palamós. Aquestes imatges són les mateixes que han vist els compradors de les llotges abans de fer la compra.

Figura 20: Imatge d'una capsa de pops blancs de la subhasta de la llotja de Llançà

Font: Obtinguda de la llotja de Llançà

En les imatges hi ha una mitja de 21,1 pops per imatge i com ja es comentarà més endavant, per a obtenir bons resultats en aquest projecte serà de gran importància la disposició dels pops en les caixes. En total s'ha utilitzat 410 imatges per crear la base de dades.

4.2 Preparació de les imatges

Per tal de poder utilitzar les imatges s'ha procedit a editat una per una. Primerament, s'ha canviat el nom per evitar que cap contingui el mateix. El nom de les imatges és: més_día_lot_númerodefoto.jpg. Totes aquestes dades han estat proporcionades per les llotges en un document Excel juntament amb el pes en kg, el preu per kg i el calibre de cada caixa. A continuació s'han editat amb el programa labelme, un programa d'anotació gràfica gratuït creat pel MIT (Massachusetts Institute of Technology). Aquest permet dibuixar polígons, sobre les imatges. Per cada cap de pop blanc s'ha dibuixat el seu contorn. El polígon resultant s'anomena màscara. Per tant, per cada imatge s'ha obtingut tantes màscares com pops blancs. Aquest és el procés d'etiquetatge.

Un cop s'ha acabat d'editar la imatge es guarda amb un fitxer amb format JSON amb el mateix nom que la imatge original. En el fitxer JSON està anotada la classe de cada màscara amb les coordenades

de tots els punts del polígon que la componen. Per tant, la base de dades està formada per 410 imatges en format JPEG i 410 fitxers JSON.

Els polígons resultants de l'edició amb labelme seran els que s'entrenaran en la Mask R-CNN i per tant els que reconeixerà el programa. Com l'objectiu d'aquest projecte és la detecció dels pops blancs s'ha decidit marcar el contorn del cap dels pops blancs de les imatges, ja que és la part del cos més senzilla d'ubicar.

La qualitat de les màscares serà un factor que influenciarà molt en els resultats que s'obtindran del model, per tant és important que les màscares que es dibuixin s'ajustin al màxim al contorn dels caps dels pops. També influenciarà molt en els resultats la disposició dels pops en les imatges. Si els pops estan ordenats i els caps es poden identificar clarament serà molt més senzill etiquetar i el model serà capaç de reconèixer els caps amb més facilitat. Això, però, no serà sempre d'aquesta manera, ja que els pops estan vius i el moviment dins la capsa farà que les imatges siguin difícils d'etiquetar i d'analitzar pel programa. En la figura 21 es pot veure un exemple d'una capsa ordenada i una desordenada.

Figura 21: Capsa ordenada (esquerra) i desordenada (dreta)

Font: Obtingudes de la llotja de Llançà

En la figura 22 es pot veure una imatge que està sent editada amb el labelme. Es pot observar les màscares pintades de vermell per sobre de la imatge i a la dreta el llistat de màscares. Cada màscara li correspon una etiqueta anomenada Pop_blanc que serà la classe que reconeixerà la Mask R-CNN. Com només es disposa d'una classe, el programa només identificarà els caps dels pops.

Figura 22: Edició d'una imatge amb labelme

Font: Realitzada pel mateix alumne

4.3 Entorn per realitzar l'entrenament i la inferència

Per tal de realitzar l'entrenament i en alguns casos la inferència dels resultats s'ha utilitzat el Google Colaboratory (Colab), un entorn gratuït de Jupyter Notebook que permet escriure i executar python des del navegador. Colab permet escriure *Notebooks* i compartir-los amb altres usuaris i utilitza com a directori Google Drive. Aquesta eina ha estat de gran utilitat per a compartir el treball amb el director del projecte i amb altres persones de l'equip. A més la principal raó d'utilitzar Google Colab és que utilitza unitats de GPU de Google i no consumeix els recursos de l'ordinador personal. Això ha fet que el temps d'entrenament es redueixi. Un altre avantatge de Colab és que ja té moltes de les llibreries de python instal·lades i ha fet que la seva utilització sigui molt més senzilla.

També s'ha utilitzat l'ordinador personal mitjançant el programa Anaconda i l'editor Spyder per fer el programa d'interferència de resultats explicat en l'apartat 5.7.

4.4 Divisió de la base de dades

Una vegada etiquetades totes les imatges amb el labelme s'han de separar en tres directoris diferents. Aquesta divisió hauria de ser aleatòria, però com que el temps que requereix editar les imatges és molt elevat i s'ha treballat amb una base de dades limitada, per tal d'optimitzar els resultats no s'ha fet la separació de manera aleatòria.

Els directoris són els següents:

- Train: Imatges utilitzades per entrenat el model. Correspon al 80% d'aquestes.
- Val o validació: Imatges utilitzades per validar el model durant l'entrenament. Ha d'estar formada per imatges molt variades on hi hagi una bona representació de la varietat de base de dades. Durant el procés d'entrenament, al final de cada *epoch* el model calcula les pèrdues de validació de la mateixa manera que calcula les pèrdues amb les imatges d'entrenament. Això permetrà saber com es comporta el model amb imatges que no ha vist abans. Correspon al 10% de les imatges.
- Test: Imatges utilitzades per mirar els resultats amb noves imatges que el model no ha vist mai. Com en la carpeta de val, ha de ser variada. Correspon al 10% de les imatges.

Els percentatges comentats seran valors orientatius i la distribució de la base de dades variarà en funció de la situació.

4.5 Entrenament

L'entrenament és el procés en el qual el model aprèn de les imatges que hi ha en carpeta train. És un procés iteratiu en el qual analitza totes les imatges i calcula els pesos. Com ja hem comentat abans, comencem l'entrenament amb els pesos de COCO. En el projecte s'han realitzat 200 *epoch* per entrenament, però com ja comentarem en l'apartat 5, s'ha estudiat els resultats per diferent número d'*epoch*.

L'entrenament comença amb un *learning rate* igual a 0,001 i quan s'han fet la meitat d'*epoch* disminueix a 0,0001.

S'ha comprovat que com més alt sigui el *batch size* utilitzat millors resultats dóna el model. El valor utilitzat per aquest paràmetre ha estat 4, que és el màxim que es pot utilitzar sense utilitzar tota la memòria disponible en Google Colab.

Per últim, es descartaran totes les *bounding boxes* amb menys del 90% de seguretat que pertanyin a la classe Pop_blanc.

A causa del fet que cada entrenament dura més de 12 hores en completar-se i la poca disponibilitat de GPU lliures en Google Colab, no s'ha estudiat en profunditat l'impacte de cada paràmetre comentat en el model.

5 Anàlisi de resultats

Un cop finalitzat el procés d'entrenament ja es poden analitzar els resultats. En total s'han realitzat cinc entrenaments diferents. També s'han realitzat entrenaments extres per comparar la diferència d'entrenar 100 o 200 *epoch*.

Per analitzar els resultats s'estudiarà el nombre de pops que detecta el model en diverses imatges i la qualitat de les màscares que genera. Mitjançant l'eina TensorBoard de TensorFlow es visualitzaran les pèrdues de validació (*val_los*) i d'entrenament (*loss*). Finalment, s'estudiarà les àrees i s'intentarà predir el calibre.

L'error s'ha calculat fent la diferència entre el nombre de pops blancs que hi ha realment en una caixa amb els que troba el model.

5.1 Entrenament 0

El primer entrenament o entrenament 0 s'ha dut a terme amb un total de 48 imatges en la capeta train i 7 a la del val i test. Totes aquestes imatges corresponen a les que es van enviar de la llotja de Llançà en el mes de febrer.

Estudi de l'error i qualitat de les màscares

S'ha calculat l'error en el nombre de pops blancs que detecta el model per imatge en les imatges de la carpeta de validació. L'error és de 2,7 pops per imatge. És bastant correcte per ser el primer entrenament i disposant d'una base de dades tan petita.

Si s'analitzen les màscares, s'observa que hi ha bastants errors en cada imatge, com per exemple, detecta alguns caps i encercla altres parts dels pops. Es pot veure clarament com en les imatges en les quals els pops estan ben col·locats el programa reconeix amb més efectivitat els caps i fa millor les màscares.

En les imatges de la figura 23 es pot veure a l'esquerra les màscares fetes a mà amb el labelme i a la dreta les que ens ha proporcionat el model.

Figura 23: Màscares fetes amb labelme (esquerra) i obtingudes amb el model de l'entrenament 0 (dreta)

Font: Realitzada pel mateix alumne

De les 14 imatges que s'han analitzat, la imatge de la figura 23 és l'única en la qual el model ha estat capaç d'encertar el nombre de pops, 22 en total. Tot i això, no detecta de forma correcta tots els caps i només reconeix 20 dels 22 pops.

En les imatges de la figura 24, el model no ha detectat 9 dels caps. El principal problema ha estat que els caps estan molt distorsionats, sent aixafats o tallats per les potes d'altres pops. Aquí es pot veure la importància que té que els pops estiguin ben col·locats.

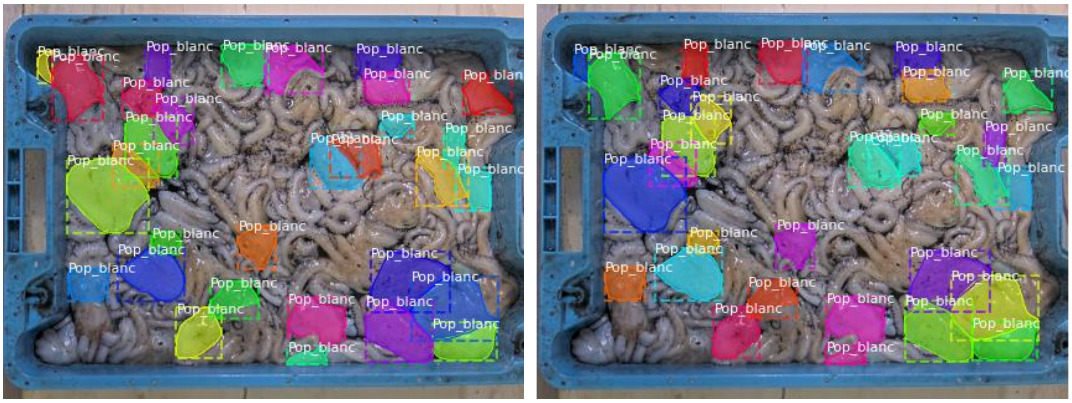


Figura 24: Màscares amb labelme (esquerra) i obtingudes de l'entrenament 0 (dreta)

Font: Realitzada pel mateix alumne

Estudi de les pèrdues

En les imatges de la figura 25 es pot veure la representació de les pèrdues d'entrenament i de validació. Com ja s'ha comentat anteriorment, les pèrdues fan referència a l'error comès comparant les màscares que crea el model amb les que s'han dibuixat amb el labelme. Aquest valor es calcula a partir d'una funció de pèrdues.

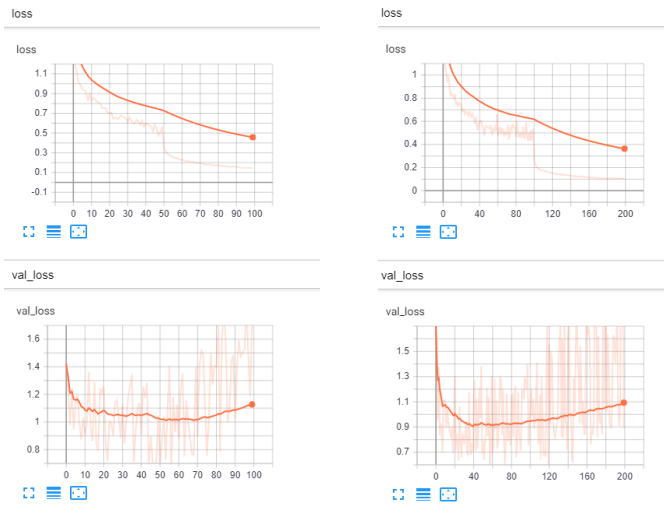


Figura 25: Pèrdues de l'entrenament 0 per 100 epoch (esquerra) i per 200 epoch (dreta)

Font: Realitzada per l'alumne a partir de la sortida del Tensorboard

En ambdues imatges es pot veure com les pèrdues d'entrenament disminueixen progressivament fins a 4,3 en el cas de 100 *epoch* i fins a 3,8 en el de 200. En canvi, les pèrdues de validació disminueixen molt menys. Primer disminueixen i quan l'entrenament arriba a un cert número d'*epoch* comencen a augmentar. També s'ha de comentar que els valors de les pèrdues de validació en cada *epoch* varien molt. Aquesta variació és deguda al fet que les dades tenen molta variabilitat i que hi ha poques imatges en la carpeta de validació. A causa de la quantitat de temps que en necessita per etiquetar les dades, en aquest i en els futurs experiments es prioritzarà augmentar la base de dades d'entrenament a la de validació.

Quan en el transcurs d'un entrenament no disminueixen les pèrdues de validació, però si les d'entrenament, es diu que el model té *overfitting.* Es produeix quan el model té un rendiment molt alt amb el conjunt de dades d'entrenament, però en canvi, té un rendiment baix amb les dades de validació o test. Això és pel fet que el model no aprèn a reconèixer l'objecte en qüestió sinó que aprèn a segmentar només les imatges d'entrenament. Es produeix normalment quan el número d'*epochs* és massa gran, el *learning rate* massa alt o el conjunt de dades és massa petit.

En la figura 26 es pot veure un exemple de casos que ens podem trobar en representar les pèrdues. Les línies blaves fan referència a les pèrdues d'entrenament i les vermelles a les de validació. En l'A les pèrdues no estan disminuint. Això implica que el model no està aprenent. En el B hi ha un exemple d'*overfitting* en el que les pèrdues de validació augmenten a partir d'un número d'*epoch*.

Aquest és bastant similar al gràfic de pèrdues obtingut en l'entrenament 0. En canvi els casos C i D són exemples d'un bon model en el qual les pèrdues disminueixen en augmentar les *epoch* i és el que es pretendrà aconseguir en el treball.

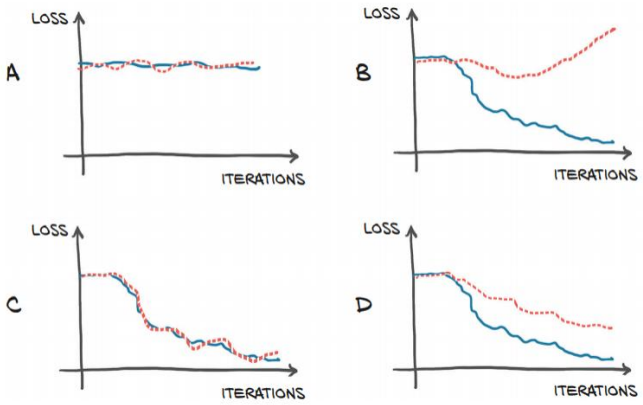


Figura 26: Corbes de pèrdues de quatre models diferents

Font: Deep Learning with PyTorch [43]

En el cas del model de l'entrenament 0, la figura 25 ens indica que té *overfitting* segurament causat perquè el conjunt de dades d'entrenament és massa petit. Tot i això, per tenir una total certesa s'hauria de comparar l'error obtingut amb imatges de la carpeta d'entrenament i de test. Si l'error amb imatges que no ha vist mai (test) és bastant superior, el model tindrà *overfitting*. Si es comparen els resultats de l'entrenament de 100 i 200 *epoch* (vegeu figura 25) es pot observar com per 200 *epoch* les pèrdues de validació són menors. Per tant, no és possible que el número d'*epoch* sigui responsable de què el model tingui *overfitting*. Tampoc el *learning rate*, ja que s'han realitzat entrenaments amb *learning rates* inferiors i les pèrdues de validació han estat superiors.

Hi ha tres maneres per disminuir l'*overfitting*:

1. Canviar la distribució dels fitxers de les carpetes val, test i train.
2. Augmentar la base de dades.
3. Utilitzar *data* augmentation per augmentar la base de dades.

El *data augmentation* consisteix a modificar les imatges de la base de dades d'entrenament amb l'objectiu de tenir més imatges per entrenar. La mateixa Mask R-CNN ja té incorporada una funció que rota les imatges. Altres *data augmentation* són funcions que disminueixen o augmenten la seva mida, canvien el seu color o les voltegen. En el cas d'aquest projecte només té sentit girar i voltejar les imatges.

5.2 Entrenament 1

En aquest entrenament s'ha utilitzat la mateixa base de dades que l'entrenament 1, però s'ha modificat la distribució de les imatges de cada carpeta. S'han fet les següents modificacions:

- S'han tret les imatges de la carpeta de test, ja que es vol el màxim d'imatges en la carpeta de train. Les imatges restants s'han distribuït en les altres dues carpetes.
- S'han escollit imatges concretes que representin la varietat d'imatges de la base de dades i s'han guardat en la carpeta de val.
- S'han afegit les imatges més complicades a la carpeta test perquè el programa aprengui a segmentar-les.

Finalment, l'entrenament s'ha dut a terme amb 49 imatges en la carpeta train i 13 en la de val.

Estudi de l'error i qualitat de les màscares

L'error amb les imatges de validació en aquest entrenament ha estat de 0,8 pops per imatge. S'ha fet una mitja per a 5 imatges.

Si s'analitzen les màscares, s'observa com el model continua tenint problemes per detectar els caps dels pops distorsionats, acostuma a confondre'ls amb altres parts del pop com les potes i no és capaç de detectar-ne bastants, sobretot aquells més distorsionats o amb potes per sobre.

Estudi de les pèrdues

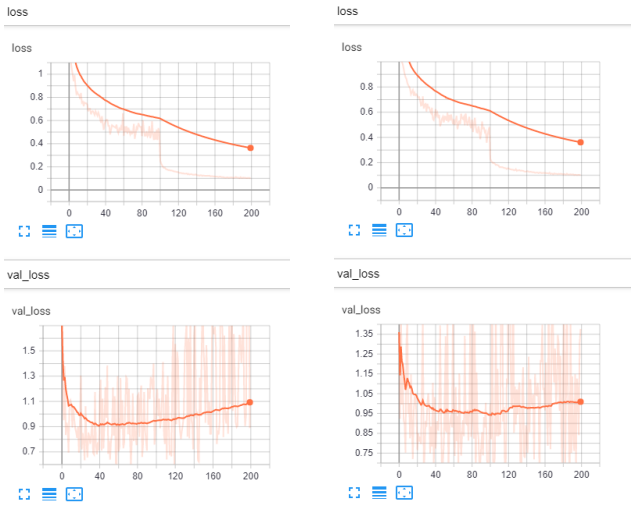


Figura 27: Pèrdues entrenament 0 (esquerra) i entrenament 1 (dreta)

Font: Realitzada per l'alumne a partir de la sortida del Tensorboard

Si es mira el gràfic de les pèrdues (vegeu la figura 27) es pot veure com les pèrdues de validació segueixen sent superiors a les d'entrenament. El problema és que segueixen havent molt poques imatges a la base de dades. Tot i això, hi ha hagut una petita disminució en les pèrdues tant de les imatges d'entrenament com de les de validació.

5.3 Entrenament 2

Per realitzar aquest entrenament s'ha augmentat la base de dades. S'han fet les següents modificacions:

- S'han etiquetat 124 imatges noves i s'han afegit a la carpeta train.
- S'ha disminuït el nombre de fotos de la carpeta val. S'han mogut 8 imatges de la carpeta val a train. Això permetrà que el model s'entreni amb el màxim nombre d'imatges.

L'entrenament s'ha dut a terme amb 181 imatges a la carpeta train i 5 a la val.

Estudi de l'error i qualitat de les màscares

S'ha calculat l'error en el nombre de pops blancs que detecta el model per imatge en les fotografies de la carpeta de validació. L'error ha estat d'1,8 pops per imatge. En un primer principi es podria pensar que el resultat ha empitjorat comparat amb l'entrenament 1, però les màscares són de millor qualitat. Les principals diferències entre les imatges obtingudes de l'entrenament 1 i 2 són les següents:

- En l'entrenament 2 detecta millor on acaben els caps.
- En l'entrenament 2 el model és capaç de detectar caps de pops amb potes per sobre.
- En l'entrenament 1 conta millor el nombre de caps a causa que fa més errors i es compensen.
- En l'entrenament 2 detecta més vegades les cues com si fossin caps.

En les imatges de la figures 28 i 29 es pot veure les màscares dels diferents entrenaments:

Figura 28: Màscares obtingudes del model de l'entrenament 0 (esquerra) i 1 (dreta)

Font: Realitzada pel mateix alumne

Figura 29: Màscares obtingudes del model de l'entrenament 2 (esquerra) i dibuixades a mà amb el labelme (dreta)

Font: Realitzada pel mateix alumne

Estudi de les pèrdues

En aquest últim entrenament els resultats obtinguts de les pèrdues no són tan bons com s'esperava (vegeu la figura 30). Si es compara amb l'obtingut en l'entrenament 1, les pèrdues de validació han disminuït a diferència de les d'entrenament, que han augmentat.

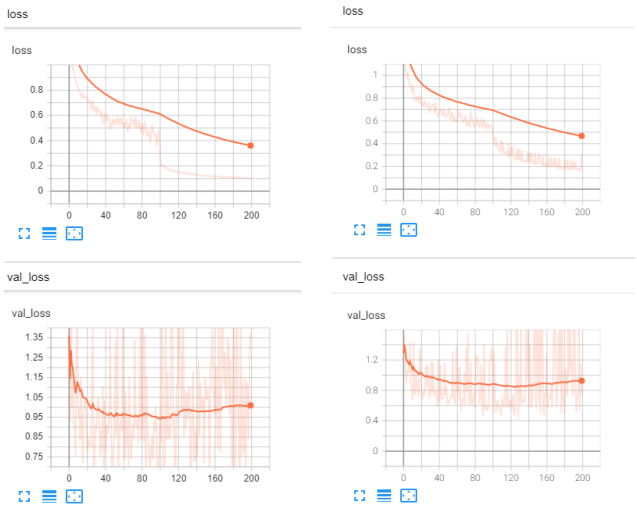


Figura 30: Pèrdues entrenament 1 (esquerra) i entrenament 2 (dreta)

Font: Realitzada per l'alumne a partir de la sortida del Tensorboard

5.4 Entrenament 3

Aquest entrenament s'ha dut a terme a partir dels pesos de l'entrenament 2, amb l'objectiu de disminuir les pèrdues i el temps d'entrenament. Es vol estudiar l'error obtingut en contar el nombre de pops blancs per imatges de la llotja de Palamós. Per tant, s'ha hagut d'etiquetar noves imatges procedents d'aquesta llotja. També s'han etiquetat imatges procedents de la llotja de Llançà.

La distribució de la base de dades és la següent:

- 305 imatges a la carpeta d'entrenament, 278 imatges de Llançà i 27 de Palamós.
- 10 imatges a la carpeta de validació, 7 de Llançà i 3 de Palamós.
- 25 imatges de la llotja de Palamós a la carpeta de test.

En aquest entrenament s'han fet 200 *epoch*.

Estudi de les pèrdues

En la figura 31 es pot veure com les pèrdues d'entrenament han disminuït fins a 0,1. Les pèrdues de validació segueixen sent bastant inferiors a les d'entrenament. Tot i això han disminuït respecte

a l'anterior entrenament. A partir de les 300 *epoch* les pèrdues de validació augmenten però no tant com en l'entrenament 2.

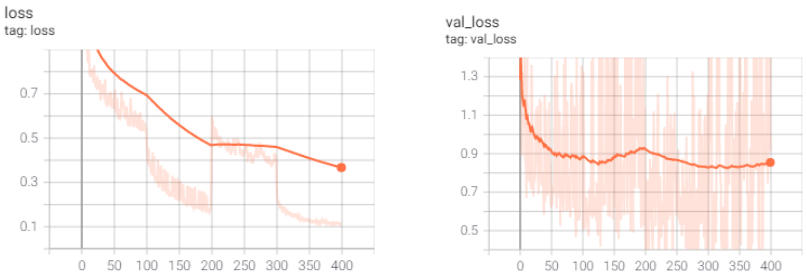


Figura 31: Pèrdues entrenament 3

Font: Realitzada per l'alumne a partir de la sortida del Tensorboard

Estudi de l'error

S'ha calculat l'error en el nombre de pops blancs que detecta el model per imatge en les imatges de la carpeta de test.

Pel fet que les pèrdues per validació són irregulars (varien molt en cada *epoch*), s'ha estudiat l'error amb els pesos de diferents *epoch*. El resultat es pot veure en la taula 1.

Epoch	*Loss*	*Val_loss*	Error mitjà
381	0,1165	0,18	2,16
386	0,1176	0,2096	1,96
400	0,1103	0,9126	1,96

Taula 1: Error mitjà amb els pesos de l'entrenament 3 amb 381, 386 i 400 epoch

Font: Realitzada pel mateix alumne

L'error mitjà és igual en el model de 400 i 386 *epoch*, però sorprenentment si es compara l'error obtingut en cada una de les 25 imatges, els resultats són lleugerament diferents.

Es podria pensar que l'entrenament no ha funcionat bé pel fet que l'error és superior a l'obtingut en l'entrenament 2. En aquest l'error ha estat d'1,8, però s'ha calculat amb les imatges de validació. Si es mesura l'error d'aquest entrenament amb les imatges de validació (conté 7 imatges de Llançà i 3 de Palamós), l'error disminueix a 1,6. Això demostra que hi ha diferència entre les imatges de Llançà i Palamós i com que la majoria de les imatges utilitzades per entrenar són de Llançà, és normal que l'error augmenti quan es calcula amb les que provenen de Palamós.

5.5 Entrenament 4

Després dels bons resultats en les pèrdues de l'entrenament 3, s'ha decidit començar l'entrenament 4 a partir dels pesos de l'entrenament 3, amb l'objectiu de minimitzar les pèrdues d'entrenament i de validació. Pel fet que en l'entrenament 3 l'error obtingut per imatges de Palamós era superior a l'obtingut per imatges procedents de Llançà, s'han etiquetat 70 noves imatges de Palamós.

La distribució de la base de dades és la següent:

- 400 imatges a la carpeta d'entrenament, 278 imatges de Llançà i 122 de Palamós.
- 10 imatges a la carpeta de validació, 7 de Llançà i 3 de Palamós.
- 60 imatges no etiquetades, 20 de Llançà i 40 de Palamós.

En aquest entrenament s'han fet 200 *epoch*.

Estudi de les pèrdues

Com era d'esperar, tant les pèrdues de validació com d'entrenament han disminuït (vegeu figura 32). El valor de les pèrdues en l'*epoch* número 600 és de 0,0912 per les d'entrenament i 0,82 per les de validació. El gràfic de pèrdues de validació segueix indicant que el model té *overfitting*.

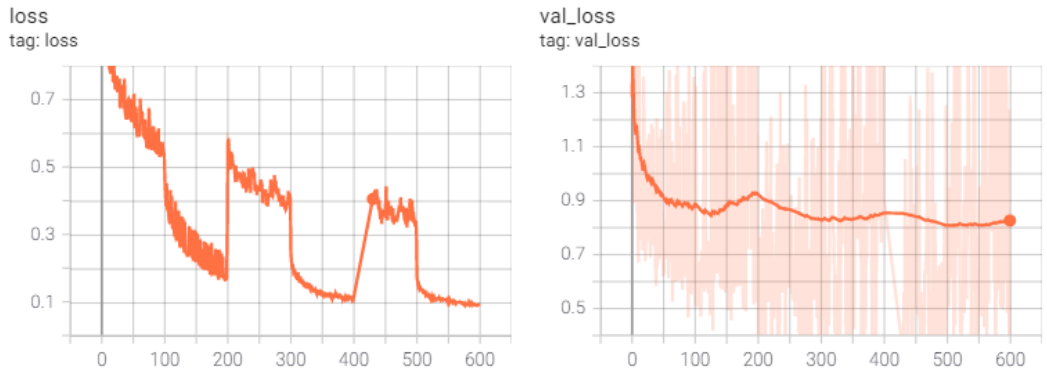


Figura 32: Pèrdues d'entrenament (esquerra) i de validació (dreta)

Font: Realitzada per l'alumne a partir de la sortida del Tensorboard

Estudi del nombre de pops i qualitat de les màscares

Amb l'objectiu d'entrenar el model amb el màxim d'imatges possibles, a diferència de la resta d'experiments, l'error s'ha calculat amb imatges no etiquetades. En anteriors entrenaments s'ha calculat l'error fent la diferència entre el nombre de màscares fetes amb el labelme i el nombre de màscares que genera el model. Per tant, no s'ha tingut en compte pops amb caps no visibles. Aquest canvi fa que el càlcul de l'error sigui més fidel a la realitat, però l'error serà major.

S'ha calculat l'error obtingut per caixa en el nombre de pops i en el pes (kg) d'aquesta. En la taula 2 es pot veure el resultat.

	Llançà	Palamós	Total
Error relatiu mitjà per caixa [%]	7,15	9,74	8,44
Error absolut mitjà [pops/caixa]	1,35	1,90	1,63
Error absolut mitjà [kg/caixa]	0,51	0,60	0,56

Taula 2: Error en el nombre de pops i pes, per imatges de test de l'entrenament 4.

Font: Realitzada per l'alumne

L'error absolut mitjà en pops/caixa ha disminuït respecte als entrenaments 2 i 3. Igual que en l'entrenament 3, el resultat de l'error per imatges de Llançà és inferior al de Palamós. Això pot ser degut al fet que el model s'ha entrenat amb més imatges de Llançà que de Palamós o que encara que s'ha intentat escollir imatges de test de Palamós i Llançà amb complexitats similars, que les imatges de Palamós ho sigui més.

Finalment, s'ha calculat l'error en pops/kg, que és el valor que interessa als compradors de les llotges. En la taula 3 es pot veure el resultat.

	Llançà	Palamós	Total
Error absolut mitjà [pops/kg]	0,020	0,170	0,095
Error absolut mitjà [pops/kg·caixa]	0,20	0,34	0,27

Taula 3: Error de pops/kg per imatges de test de l'entrenament 4.

Font: Realitzada per l'alumne

A continuació s'analitzaran les màscares generades pel model en algunes de les imatges de test:

S'ha de comentar que en moltes de les imatges hi ha pops a sobre d'altres i és difícil determinar amb seguretat el nombre de pops reals que hi ha en cada imatge. En caixes molt desordenades el

model sol trobar pops que no s'havien considerat i que és difícil determinar qui ha comès l'error. La caixa de la figura 33 és un bon exemple. El model ha detectat els dos pops encerclats que no s'havien considerat.

Figura 33: Imatge original (esquerra) i màscares obtingudes pel model (dreta)

Font: Realitzada per l'alumne

En la imatge de la figura 34, s'ha obtingut un error de 2 pops blancs. Com era esperable, els pops 5 i 21, que estan girats no han estat detectats pel model. Tampoc ha detectat el número 2 i 23, que tenen la meitat del cap ocult. Ha detectat el 19 dos cops, aquest és error freqüent sobretot en pops grans amb potes molt juntes. No ha detectat el pop número 18 i ha detectat com a pop, unes potes del pop número 24. Finalment, el model ha creat la màscara rosa a la dreta del pop 24 que no s'havia considerat com un pop blanc i que pot ser que sí que sigui un pop.

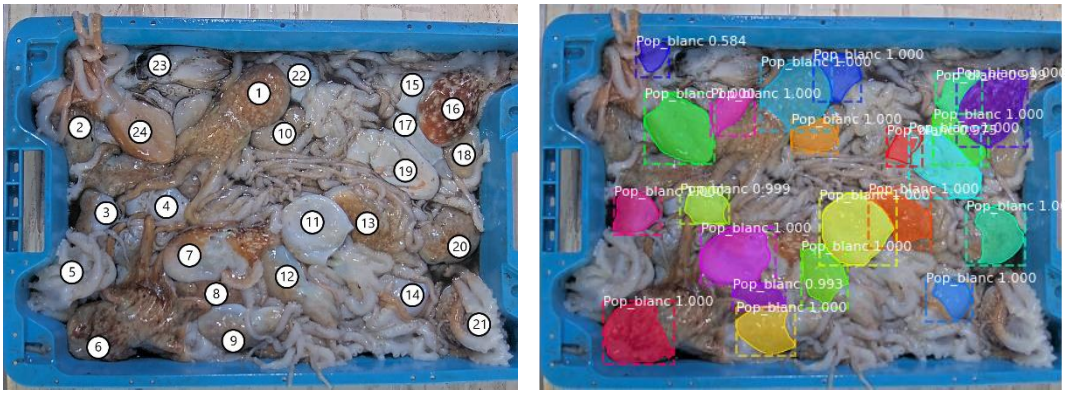


Figura 34: Imatge original (esquerra) i màscares obtingudes pel model (dreta)

Font: Realitzada per l'alumne

Algunes de les capses, com la de la figura 35 contenen aigua que dificulta la visibilitat dels caps. Com es pot veure en la figura, això provoca que no es detectin els tres caps encerclats en verd. A més, en aquesta imatge el model reconeix com a pop una pota del pop número 1.

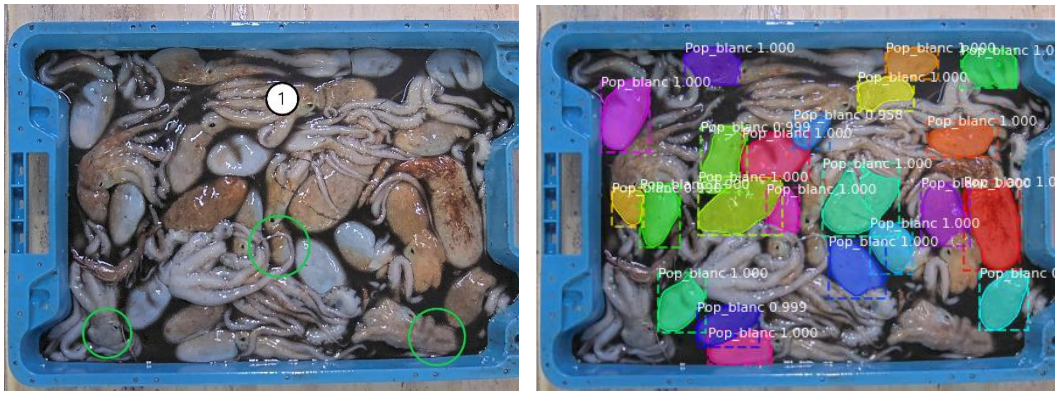


Figura 35: Imatge original (esquerra) i màscares obtingudes pel model (dreta)

Font: Realitzada per l'alumne

En la capsa de la figura 36, s'ha comès un error de 2 pops. El problema ha estat que el model ha detectat els dos caps dels pops 1 i 2 i també ha considerat que part de les seves potes com a un altre pop.

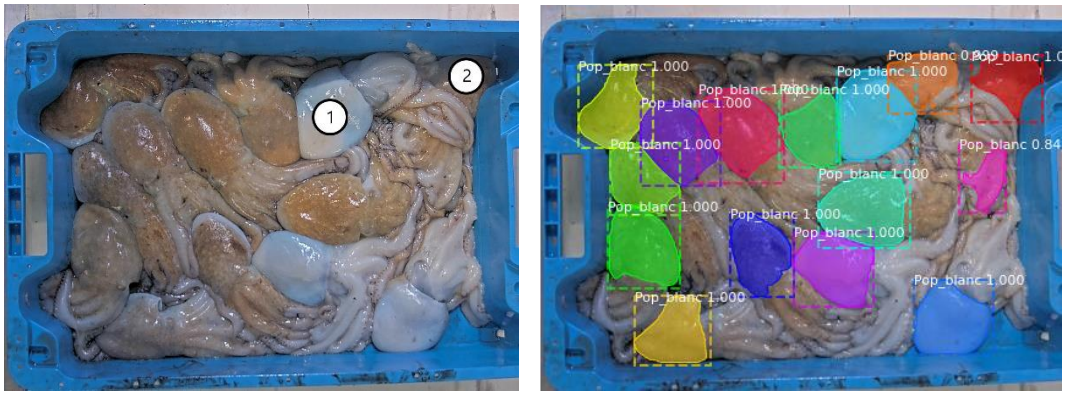


Figura 36: Imatge original (esquerra) i màscares obtingudes pel model (dreta)

Font: Realitzada per l'alumne

Un altre error típic que fa el model, és que etiqueta dues vegades els caps dels pops que tenen una pota per sobre que el separa en dos. Aquest error és provocat perquè en el moment d'etiquetar les imatges es va decidir que en aquests casos només s'etiquetaria la part del cap més gran. A causa del poc nombre d'imatges i pops amb aquesta característica, el model encara no ha après i etiqueta tot el que s'assembla a un cap encara que sigui el mateix pop. En la figura 37 es pot veure un exemple en el pop situat a la part inferior dreta de caixa.

Figura 37: Imatge original (esquerra) i màscares obtingudes pel model (dreta)

Font: Realitzada per l'alumne

Com s'ha pogut veure en aquests exemples, el model té les seves limitacions i no pot detectar pops amb caps no visibles. Tampoc dona bons resultats amb capses plenes d'aigua. Quan el model dóna millor resultats és quan els pops estan col·locats en la caixa de forma ordenada i col·locant els caps dels pops per sobre de les potes dels pops adjacents. D'aquesta manera, s'evita que hi hagi potes per sobre dels caps, per tant no es detectaran dues vegades el mateix pop i s'evitarà que no el pugui detectar. Si només es veu el cap, també s'evita que el model reconegui altres parts del pop amb formes ovalades com a caps. En la figura 38 es pot veure un exemple de caixa ben col·locada i com el model ha comès un error en el pop encerclat perquè estava tapat per una pota.

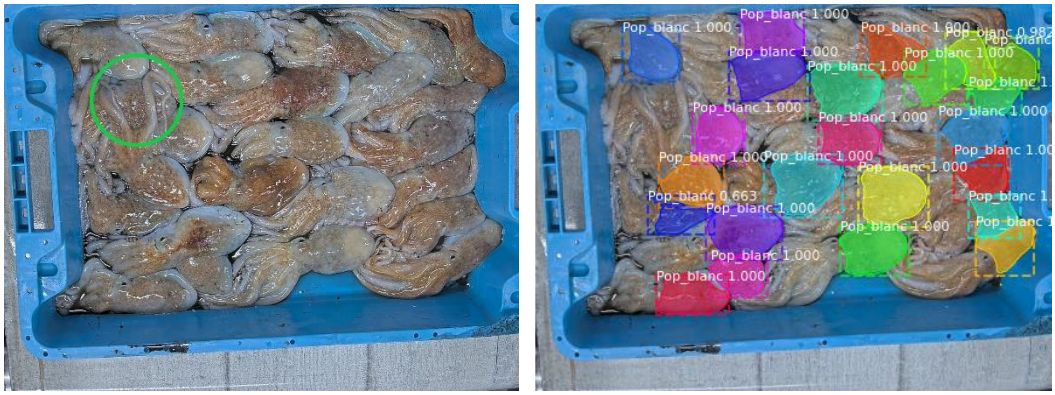


Figura 38: Imatge original (esquerra) i màscares obtingudes pel model (dreta)

Font: Realitzada per l'alumne

Per últim s'ha calculat l'error obtingut per 60 imatges aleatòries utilitzades per entrenar i s'ha comparat amb l'obtingut amb les imatges de test que el model no ha vist mai. Si l'error obtingut amb les imatges de test és molt superior, significarà que el model està aprenent a reconèixer només les imatges d'entrenament i per tant el model tindrà *overfitting*. En la taula 4 es pot veure el resultat.

Directori	Error absolut mitjà [pops/caixa]
Entrenament	0,083
Test	1,625

Taula 4: Mitjana de l'error absolut per imatges de test i train de l'entrenament 4.

Font: Realitzada per l'alumne

Efectivament, com s'ha intuït en veure tots els diagrames de pèrdues de tots els entrenaments, el model té overfitting. Tot i això l'error obtingut per imatges de test és prou bo.

5.6 Estudi de les àrees i classificació del calibre

L'objectiu d'aquest apartat és buscar una forma de predir el calibre de la caixa a partir de les àrees dels caps i la longitud d'aquests.

Tots els càlculs que s'han realitzat en aquesta secció s'han fet a partir de les màscares creades a mà amb el labelme pel fet que es necessiten les dades més exactes possibles. En total s'han utilitzat dades de 248 imatges de Llançà.

Estudi de la distribució d'àrees

En l'histograma de la figura 39 estan representades les àrees dels caps dels pops blancs de les 248 imatges. Es pot observar com segueixen una distribució weibull.

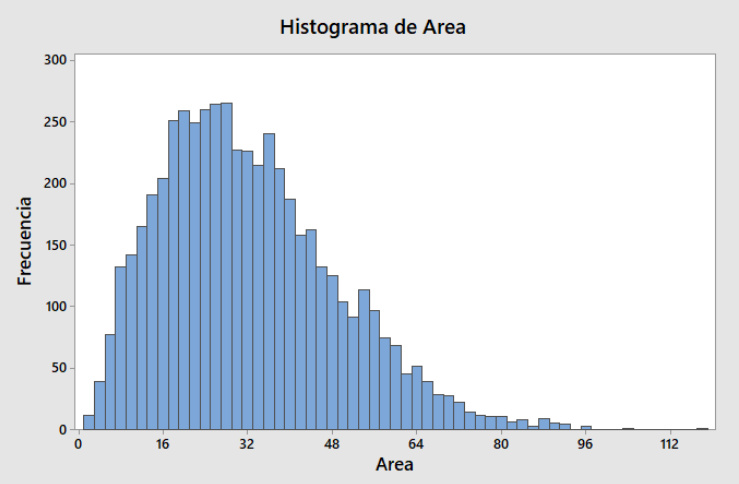


Figura 39: Histograma d'àrees

Font: Realitzada pel mateix alumne

Estudi del calibre en funció de l'àrea del cap

En l'histograma de la figura 40 es pot veure clarament com els pops de les caixes de calibre 3 i 4 no tenen gaire diferència de mida.

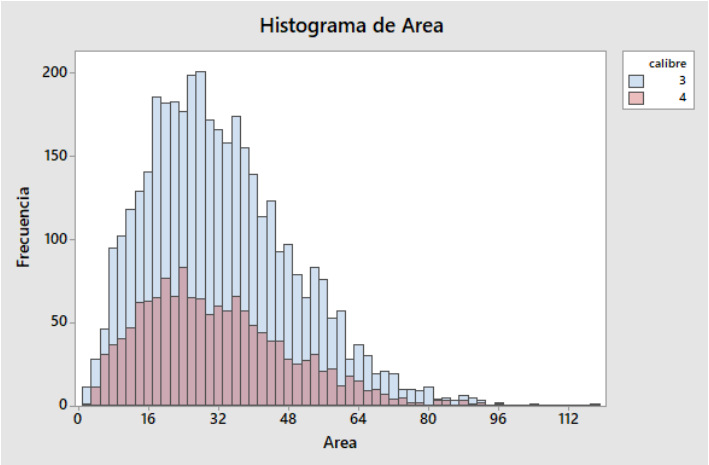


Figura 40: Histograma d'àrees classificat per calibres

Font: Realitzada pel mateix alumne

Si fem la mitja pels 2 calibres, obtenim que pel calibre 3, la mitja de l'àrea dels pops és de 32,658 cm^2, i pels de calibre 4, 31,606 cm^2. Això es pot veure més clarament en el diagrama de caixes de la figura 41.

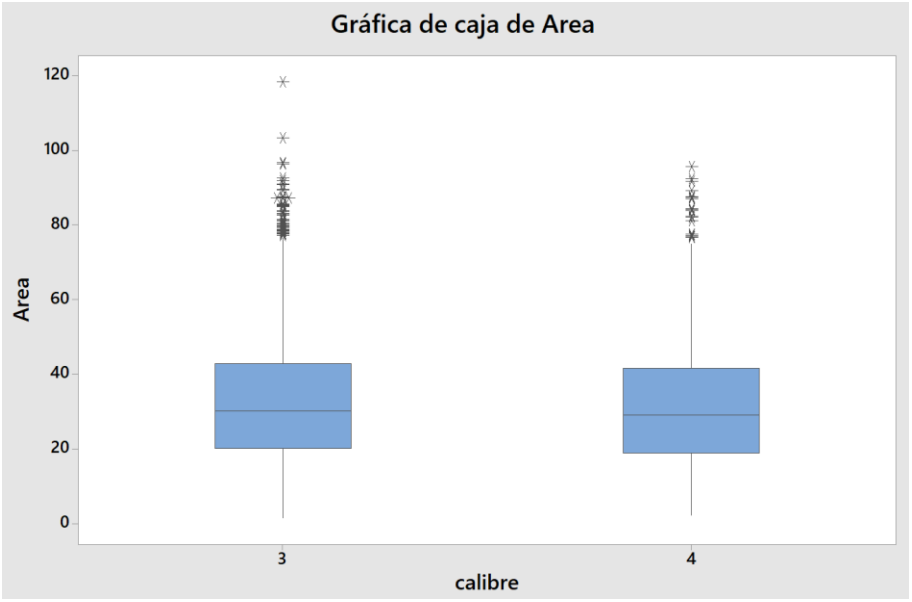


Figura 41: Diagrama de caixes d'àrees classificat per calibre

Font: Realitzada pel mateix alumne

Amb l'objectiu de seguir buscant la manera de predir el calibre amb les àrees dels pops, s'ha fet el diagrama de caixes de la figura 42 on s'ha estudiat quina influencia té l'àrea mitjana, l'àrea màxima, l'àrea mínima i el nombre de pops sobre el calibre.

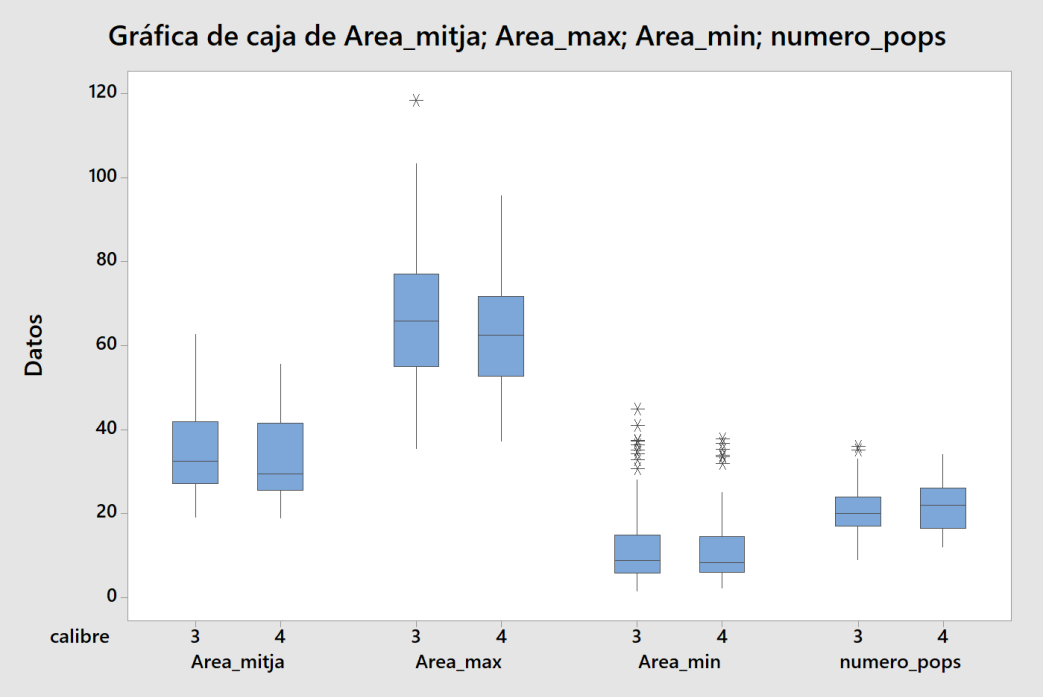


Figura 42: Diagrama de caixes d'àrees classificat per calibres

Font: Realitzada pel mateix alumne

Es pot observar com per les caixes de calibre 3 tots els diagrames de caixes estan més amunt (àrea major) i que el nombre de pops és major en les caixes de calibre 4. Per tant, sí que hi ha una influència entre aquestes variables i el calibre dels pops.

Un altre forma que s'ha utilitzat per visualitzar la influència d'aquests factors en el calibre de les caixes ha estat l'anàlisi de components principals o PCA. És una tècnica que permet descriure un conjunt de dades a partir de noves variables anomenades components.

En la figura 43 està representada la influència de cada un dels components en el calibre.

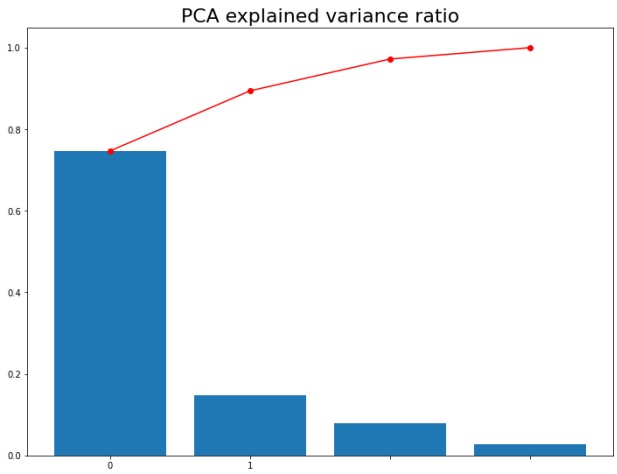


Figura 43: Principals components i variància (PCA)

Font: Realitzada pel mateix alumne

Està clar que el primer component és el que té més influència en la classificació del calibre

En la taula 5 es pot veure numèricament aquesta influència:

	Area_mitja	Area_max	Area_min	numero_pops
PC1	0.629943	0.358527	0.474053	-0.499904
PC2	-0.011388	0.787839	-0.614921	-0.032440
PC3	0.004395	0.414067	0.489954	0.767121
PC4	-0.776545	0.281632	0.396348	-0.400711

Taula 5: Influència de l'àrea i el nombre de pops en els components principals

Font: Realitzada pel mateix alumne

La component 1 es pot escriure com:

$$\text{PC1} = 0{,}699 \cdot \text{Area\_mitja} + 0{,}358 \cdot \text{Area\_max} + 0{,}474 \cdot \text{Area\_min} - 0{,}4999 \cdot \text{numero\_pops}$$

Podem veure com l'àrea mitjana és l'atribut que influencia més en la classificació. Seguit del nombre de pops, l'àrea mínima i l'àrea màxima.

Per intentar fer la classificació s'ha utilitzat l'algoritme de *Neraest Neighbors*. Aquest algoritme classifica en funció de la distància que els punts tenen amb els seus veïns.

El resultat es pot veure en la taula 6.

		Predicció	
		3	4
Real	3	57	5
	4	17	4

Taula 6: Resultats classificació amb Neraest Neighbors

Font: Realitzada pel mateix alumne

Els resultats són per a 85 imatges. D'aquestes imatges, 62 eren realment corresponents al calibre 3 i n'ha encertat 57 i 21 eren del calibre 4 i n'ha encertat 4. S'ha obtingut una precisió de 0,73.

Aquests resultats són molt dolents pel fet que l'algoritme no és capaç de classificar les caixes de calibre 4.

Com s'ha vist amb els histogrames i els diagrames de caixes, això pot haver estat provocat perquè no hi ha gaire diferència en les àrees dels pops blancs en les caixes de calibre 3 i 4. A més, és possible que no hi hagin suficients dades pel bon funcionament de l'algoritme.

Estudi del calibre en funció de la longitud del cap

A causa que els pops en les caixes estan vius en el moment de fer les fotos, molts dels seus caps estan tapats o aixafats. Això fa que agafar l'àrea com a mesura per predir el calibre pugui generar, com s'ha pogut veure, bastants errors. Per això s'ha intentat predir el calibre a partir de la longitud del cap. S'ha elegit la longitud i no l'amplada perquè en molts dels casos, com per exemple, en la imatge de la figura 44, la longitud dels pops de la part inferior esquerra no està distorsionada i s'ha pensat que pot donar millor resultats. En canvi, molts dels caps estan aixafats o tapats i no s'aconsegueix veure l'amplada.

Figura 44: Capsa de pops blancs amb màscara

Font: Realitzada pel mateix alumne

Aquest estudi s'ha efectuat mitjançant la llibreria skeletonize de scikit-image [44], la qual et permet dibuixar la línia mitjana dintre d'un contorn.

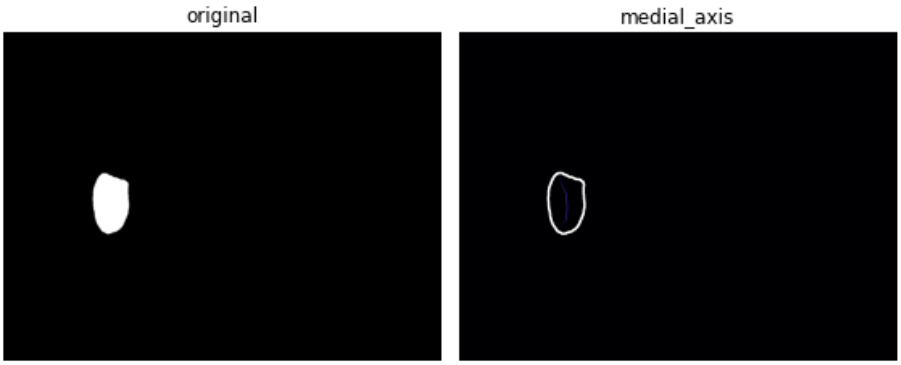


Figura 45: Màscara editada amb skeletonize (bon resultat)

Font: Realitzada pel mateix alumne

S'ha aplicat per a diferents màscares i els resultats no són gaire bons. Per a màscares estretes extreu resultats satisfactoris, però per a màscares més esfèriques la línia que dibuixa es ramifica i no permet obtenir la llargada de la màscara. En les figures 45 i 46 es pot veure el resultat en algunes màscares.

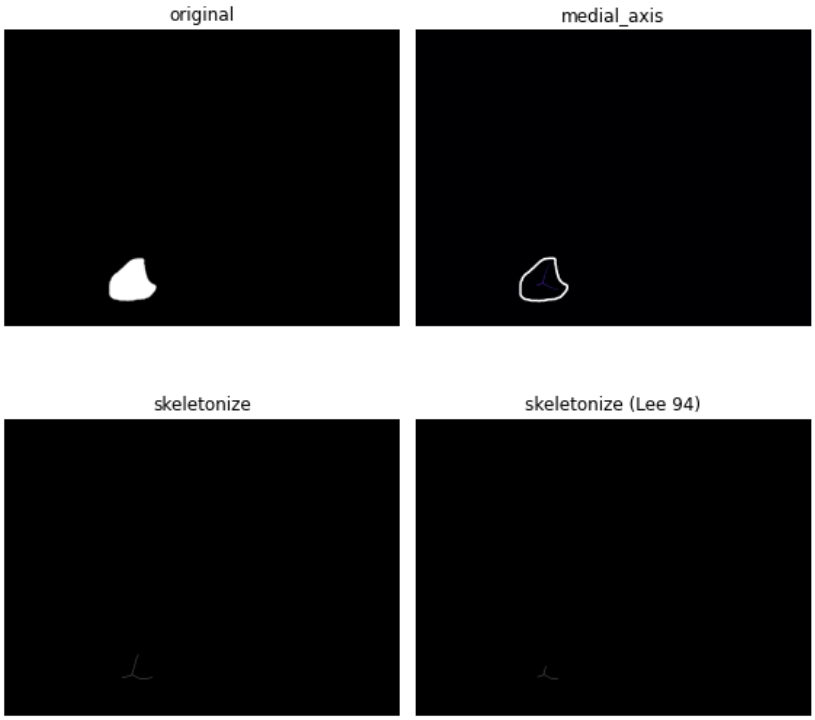


Figura 46: Màscara editada amb skeletonize (resultat dolent)

Font: Realitzada pel mateix alumne

5.7 Creació del programa d'inferència

En aquest apartat s'explicarà el funcionament del programa d'inferència de resultats. Aquest programa està pensat per ser utilitzat pels treballadors de les llotges en el cas que en un futur s'implementés el detector.

El programa utilitza funcions dels fitxers del repositori de GitHub, *Mask R-CNN for Object Detection and Segmentation* [1]. S'ha modificat el contingut del fitxer visualize.py d'aquest directori.

Per fer servir el programa primerament s'ha d'executar en la carpeta on estigui descarregada la implementació del repositori de GitHub. Es faran els imports de totes les llibreries necessàries, es definiran els directoris necessaris, es definirà les classes Pop_blanc_dataDataset i Pop_blancConfig i per últim es definirà la funció d'inferència, Pop_blancInf.

A continuació es pot executar tantes vegades com calgui la funció Pop_blancInf.

La funció Pop_blancInf conté quatre arguments:

- nom_fitxer: Nom de la imatge, ex: '02_11_30123_137.jpg'
- mode (opcional): Si no s'introdueix l'argument mode, per defecte mode = 0.
    - mode = 0, es guarda la imatge que retorna la funció i s'obre una finestra per visualitzar-la.
    - mode = 1, s'obre una finestra per visualitzar la imatge que retorna la funció.
- name (opcional): Nom amb el qual es guardarà la imatge que retorna la funció en el cas que mode = 0. Si no s'introdueix l'argument name, per defecte name = 'result'.
- data_name (opcional): Nom del fitxer, en format csv, que conté el nom de les imatges i el pes en kg de la caixa. Si no s'introdueix l'argument data_name, per defecte data_name = 'dades.csv'.

La funció Pop_blancInf retorna la imatge original i per cada pop blanc que detecta dibuixa la silueta de la màscara (no la pinta perquè així els pescadors poden veure el color del pop blanc i decidir si està en bon estat) i la *bounding box.* En la part superior de la imatge s'indica el nombre de pops que s'han detectat en la imatge, i en la inferior el nombre de pops/kg de la caixa. En la figura 47 es pot veure un exemple.

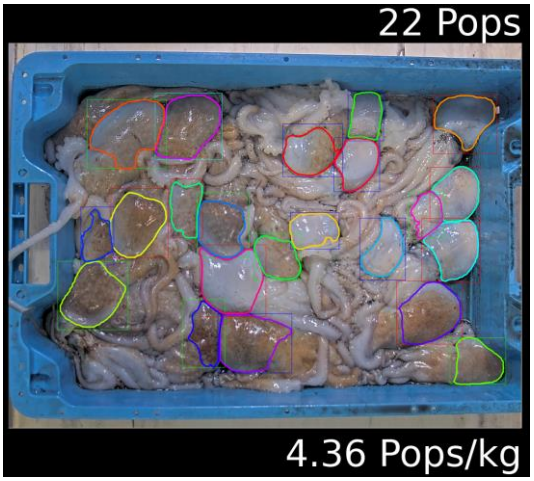


Figura 47: Output de la funció Pop_blancInf

Font: Realitzat pel mateix alumne

6 Planificació temporal

S'ha realitzat un diagrama de Gantt, representat en la figura 48 per planificar el temps que s'ha dedicat en cada fase del projecte. Aquest projecte s'ha dut a terme durant el quadrimestre de primavera de 2021. Es va començar el dia 7 de febrer i s'ha acabat el dia 21 de juny.

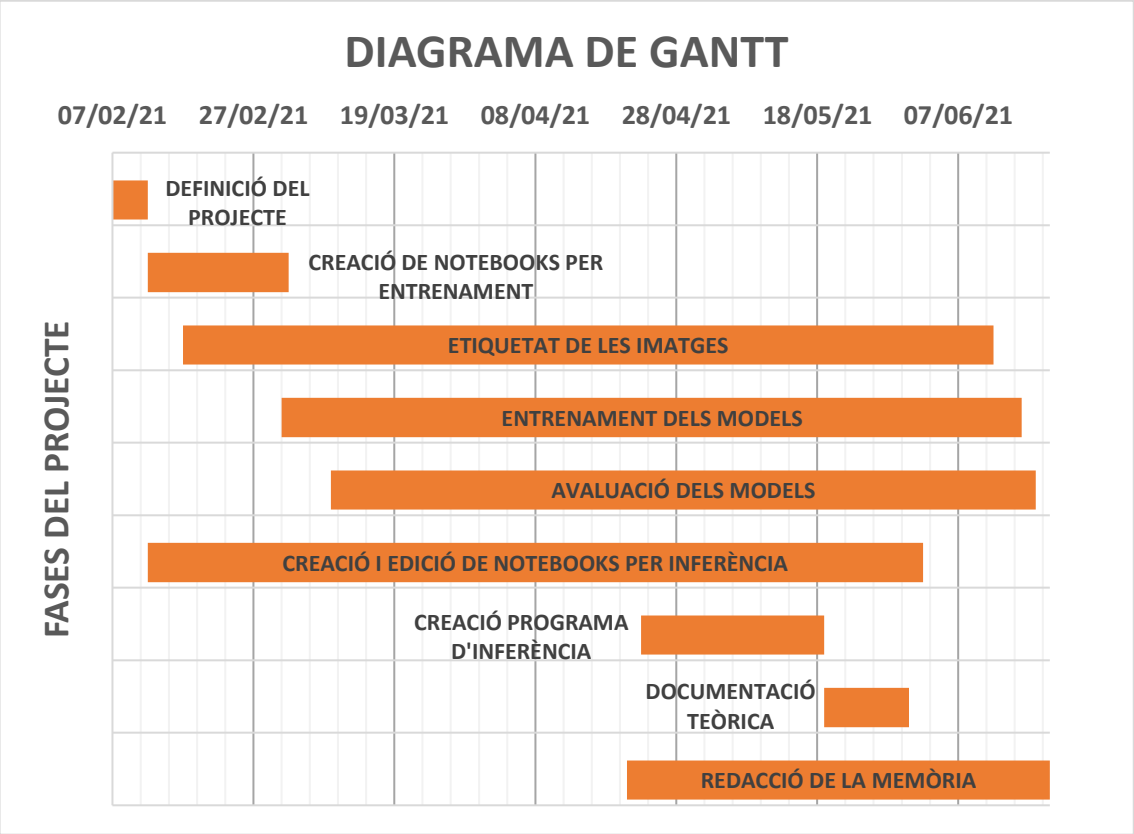


Figura 48: Diagrama de Gantt del projecte

Font: Realitzat pel mateix alumne

7 Estudi econòmic

En aquest apartat s'explicaran els costos de la realització del projecte i s'estimarà un pressupost.

Els costos de mà d'obra estan conformats per les hores de feina de l'autor del treball i del director. Es poden calcular a partir de l'equació 13. El sou/hora de l'autor s'ha establert com el sou mínim d'un conveni de pràctiques realitzat per un estudiant de l'escola (ETSEIB), 8 €/h i el del director és el sou d'un enginyer sènior, 60 €/h. Les hores treballades de l'autor s'ha estimat a 300. Aquest càlcul s'ha fet tenint en compte que aquest projecte equival a 12 ETCS i que cada ETCS equival a 25 h. Les hores treballades pel director s'han estimat a 40. Per tant, s'obté que el cost de mà d'obra tant de l'autor com del director és de 2.400 €.

$$cost\ mà\ d'obra = hores \cdot sou/hora \quad (13)$$

Equació 13: *Càlcul dels costos de mà d'obra*

S'ha considerat el cost dels aparells electrònics utilitzats durant el treball. Aquests aparells han estat un ordinador de sobre taula, un teclat, un ratolí, una pantalla i uns auriculars. S'han calculat els costos d'amortització d'aquests aparells a partir de l'equació 14. El coeficient lineal màxim segons l'Agència Tributària espanyola per equips per processar informació és del 25% per un any [45]. S'ha calculat aquest coeficient per 134 dies, que és el temps que s'ha tardat a fer el treball i equival a 9,1%. El cost de l'equip informàtic és de 1.352 €. Per tant, el cost d'amortització és de 124,09 €.

$$cost\ amortització = coeficient\ lienal\ màxim \cdot cost\ \text{equip informàtic} \quad (14)$$

Equació 14: *Cost de l'amortització de l'equip informàtic*

No s'han considerat costos de cap software pel fet que s'han emprat Google Colaboratory, python 3 i labelme, que són gratuïts. L'únic programa utilitzat ha estat el Minitab, el qual s'usa per a l'anàlisi de dades i l'escola (ETSEIB) proporciona accés gratuït.

S'ha considerat els costos indirectes de l'electricitat i l'internet consumits per l'ordinador. S'ha de mencionar que l'ús de Google Colab ha permès disminuir molt el consum energètic del treball. Aquests costos s'han considerat com un 7% dels costos directes del treball pel fet que són difícils de calcular. S'ha establert un 7% sobre els costos directes tal com va aplicar Ricard Calvo en el seu TFG [46] amb despeses molt similars a aquest. El total és de 344,7 €.

Finalment, s'ha destinat una part del pressupost a possibles imprevistos que puguin succeir durant la realització del treball. Aquest s'ha d'incloure per actuar en el cas que s'espatlli algun dels aparells informàtics ja comentats o per si Google Colab dona problemes i es decideix utilitzar un altre servei de GPU no gratuït. Aquest cost és de 400 €.

En la taula 7 s'ha calculat el pressupost final.

Tipus de Cost	Concepte		Cost (€)
Directe	Mà d'obra	Autor	2.400
		Director	2.400
	Amortització		124
Indirecte	Llum i internet		345
Imprevistos			400
Total			**5.669**

Taula 7: Costos del projecte

Font: Realitzat pel mateix alumne

8 Impacte ambiental

La creació del detector de contorns no ha suposat cap impacte ambiental positiu i si en un futur s'implementés el detector en el sistema de subhastes de les llotges, no suposaria cap impacte positiu ni negatiu.

L'ús d'un ordinador i una pantalla han comportat un consum energètic. També tots els processos relacionats amb la creació dels dispositius informàtics utilitzats en el treball van tenir en el seu dia un impacte ambiental negatiu.

En conclusió, l'impacte ambiental d'aquest projecte és molt baix.

Conclusions

L'objectiu d'aquest projecte és realitzar un sistema de segmentació semàntica capaç de detectar, contar i classificar per calibre pops blancs de les caixes provinents de les llotges de pescadors de Llançà i Palamós.

L'objectiu de detecció es dóna com a assolit gràcies als resultats obtinguts del detector de contorns. En cada entrenament les màscares han estat de major qualitat. El comptatge de pops blancs s'ha assolit amb un error mitjà d'1,63 pops per caixa i l'error obtingut en pops/kg és molt inferior a la unitat. Considero que aquests resultats permetrien una futura implementació d'aquest detector en les llotges.

Els resultats obtinguts indiquen que en el cas de la implementació del detector en les llotges, seria molt important per obtenir uns bons resultats, que els pops estiguessin ben col·locats en les caixes.

No s'ha pogut classificar els pops per calibre pel fet que no hi ha una diferència significativa entre la mida dels pops de calibre 3 i 4.

El model presenta un clar cas d'*overfitting* que ha anat disminuint en cada entrenament. Aquest fenomen podria disminuir molt amb l'addició de més dades d'entrenament i ajustant els paràmetres del model.

L'experiència amb l'entorn de Google Collaboratory per realitzar els entrenaments ha estat positiva. Tot i això, treballar amb Colab també té desavantatges, com el fet que moltes vegades no hi hagi cap GPU lliure o que es desconnecti de la GPU durant l'entrenament.

Agraïments

Principalment vull donar les gràcies al professor Vicenç Parisi, director del projecte, per donar-me l'oportunitat de fer aquest treball i per la seva dedicació.

També estic molt agraït al Sergi Selles i al Gabriel Barios, que han fet treballs similars al meu per ajudar-me i resoldre els meus dubtes.

Bibliografia

Referències bibliogràfiques

[1] w. cclauss, «Mask R-CNN for Object Detection and Segmentation,» 1 Abril 2019. [En línea]. Available: https://github.com/matterport/Mask_RCNN. [Último acceso: 30 Maig 2021].

[2] R. Demush, «A Brief History of Computer Vision (and Convolutional Neural Networks),» 2019. [En línea]. Available: https://hackernoon.com/a-brief-history-of-computer-vision-and-convolutional-neural-networks-8fe8aacc79f3. [Último acceso: 2021 Maig 28].

[3] D. H. H. i. T. N. Wiesel, «Receptive fields of single neurones in the cat's striate cortex,» The Journal of Physiology, 1959.

[4] D. Marr, «Vision: A Computational Investigation into the Human Representation and Processing of Visual Information,» 1982.

[5] IMAGENET, «ImageNet Large Scale Visual Recognition Challenge (ILSVRC),» 2020. [En línea]. Available: https://www.image-net.org/challenges/LSVRC/. [Último acceso: 28 Maig 2021].

[6] H. P. J. E. i. J. M. Hugo Mayo, «History of Machine Learning,» AI in Radiology, 2018. [En línea]. Available: https://www.doc.ic.ac.uk/~jce317/history-machine-learning.html#top. [Último acceso: 28 Maig 2021].

[7] viso.ai, «56 Most Popular Computer Vision Applications in 2021,» 2021. [En línea]. Available: https://viso.ai/applications/computer-vision-applications/. [Último acceso: 28 Maig 2021].

[8] I. Mihajlovic, «Everything You Ever Wanted To Know About Computer Vision,» Towards Data Science, 25 Abril 2019. [En línea]. Available: https://towardsdatascience.com/everything-you-ever-wanted-to-know-about-computer-vision-heres-a-look-why-it-s-so-awesome-e8a58dfb641e. [Último acceso: 24 Maig 2021].

[9] H. Lamba, «Understanding Semantic Segmentation with UNET,» Towards Data Science, 17 Febrer 2019. [En línea]. Available: https://towardsdatascience.com/understanding-semantic-segmentation-with-unet-6be4f42d4b47. [Último acceso: 24 Maig 2021].

[10] DEEPLIZARD, «Deep Learning And Artificial Neural Networks For Beginners,» 17 Novembre 2017. [En línea]. Available: https://deeplizard.com/learn/video/gZmobeGL0Yg. [Último acceso: 25 Maig 2021].

[11] deeplizard, «Unsupervised Learning In Machine Learning,» DEEPLIZARD, 2 Desembre 2017. [En línea]. Available: https://deeplizard.com/learn/video/lEfrr0Yr684. [Último acceso: 24 Maig 2021].

[12] A. M. Aditya Sharma, «Understanding Autoencoders using Tensorflow (Python),» Learn OpenCV, 15 Novembre 2017. [En línea]. Available: https://learnopencv.com/tag/denoising/. [Último acceso: 24 Maig 2021].

[13] deeplizard, «Semi-Supervised Learning For Machine Learning,» DEEPLIZARD, 3 Desembre 2017. [En línea]. Available: https://deeplizard.com/learn/video/b-yhKUINb7o. [Último acceso: 24 Maig 2021].

[14] K. B. Błażej Osiński, «What is reinforcement learning? The complete guide,» deepsense.ai, 5 Juliol 2018. [En línea]. Available: https://deepsense.ai/what-is-reinforcement-learning-the-complete-guide/. [Último acceso: 24 Maig 2021].

[15] deeplizard, «Deep Learning Explained,» DEEPLIZARD, 22 Novembre 2017. [En línea]. Available: https://deeplizard.com/learn/video/OT1jslLoCyA. [Último acceso: 26 Maig 2021].

[16] «Artificial Neural Networks Explained,» DEEPLIZARD, 22 Novembre 2017. [En línea]. Available: https://deeplizard.com/learn/video/hfK_dvC-avg. [Último acceso: 26 Maig 2021].

[17] deeplizard, «Layers In A Neural Network Explained,» DEEPLIZARD, 22 Novembre 2017. [En línea]. Available: https://deeplizard.com/learn/video/FK77zZxaBoI. [Último acceso: 26 Maig 2021].

[18] D. Asanka, «Implement Artificial Neural Networks (ANNs) in SQL Server,» SQLShack, 14 Abril 2020. [En línea]. Available: https://www.sqlshack.com/implement-artificial-neural-networks-anns-in-sql-server/. [Último acceso: 26 Maig 2021].

[19] deeplizard, «Activation Functions In A Neural Network Explained,» DEEPLIZARD, 22 Novembre 2017. [En línea]. Available: https://deeplizard.com/learn/video/m0pIlLfpXWE. [Último acceso: 26 Maig 2021].

[20] H. H. Sultan, «FIGURE 7 - uploaded by Hossam H. Sultan,» ResearchGate, Maig 2019. [En línea]. Available: https://www.researchgate.net/figure/ReLU-activation-function_fig7_333411007. [Último acceso: 26 Maig 2021].

[21] S. Saxena, «Introduction to Softmax for Neural Network,» Analytics Vidhya, 5 Abril 2021. [En línea]. Available: https://www.analyticsvidhya.com/blog/2021/04/introduction-to-softmax-for-neural-network/. [Último acceso: 26 Maig 2021].

[22] J. Brownlee, «How to Choose an Activation Function for Deep Learning,» Machine Learning Mastery, Gener 18 2021. [En línea]. Available: https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/. [Último acceso: 26 Maig 2021].

[23] deeplizard, «Training A Neural Network Explained,» DEEPLIZARD, 22 Novembre 2017. [En línea]. Available: https://deeplizard.com/learn/video/sZAlS3_dnk0. [Último acceso: 27 Maig 2021].

[24] deeplizard, «Loss In A Neural Network Explained,» DEEPLIZARD, 22 Novembre 2017. [En línea]. Available: https://deeplizard.com/learn/video/Skc8nqJirJg. [Último acceso: 27 Maig 2021].

[25] PyTorch, «SMOOTHL1LOSS,» 2019. [En línea]. Available: https://pytorch.org/docs/stable/generated/torch.nn.SmoothL1Loss.html. [Último acceso: 27 Maig 2021].

[26] deeplizard, «Learning Rate In A Neural Network Explained,» DEEPLIZARD, 22 Novembre 2017. [En línea]. Available: https://deeplizard.com/learn/video/jWT-AX9677k. [Último acceso: 27 Maig 2021].

[27] MkDocs, «Gradient Descent and Stochastic Gradient Descent,» [En línea]. Available: http://rasbt.github.io/mlxtend/user_guide/general_concepts/gradient-optimization/. [Último acceso: 27 Maig 2021].

[28] J. Jordan, «Setting the learning rate of your neural network,» 1 Març 2018. [En línea]. Available: https://www.jeremyjordan.me/nn-learning-rate/. [Último acceso: 27 Maig 2021].

[29] deeplizard, «Bias In An Artificial Neural Network Explained | How Bias Impacts Training,» DEEPLIZARD, 18 Abril 2018. [En línea]. Available: https://deeplizard.com/learn/video/HetFihsXSys. [Último acceso: 31 Maig 2021].

[30] deeplizard, «Convolutional Neural Networks (CNNs) Explained,» DEEPLIZARD, 9 Desembre 2017. [En línea]. Available: https://deeplizard.com/learn/video/YRhxdVk_sIs. [Último acceso: 28 Maig 2021].

[31] S. Saha, «A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way,» Towards Data Science, 15 Desembre 2018. [En línea]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. [Último acceso: 28 Maig 2021].

[32] deeplizard, «Zero Padding In Convolutional Neural Networks Explained,» DEEPLIZARD, 14 Febrer 2018. [En línea]. Available: https://deeplizard.com/learn/video/qSTv_m-KFk0. [Último acceso: 28 Maig 2021].

[33] deeplizard, «DEEPLIZARD Demo,» DEEPLIZARD, [En línea]. Available: https://deeplizard.com/resource/pavq7noze2. [Último acceso: 28 Maig 2021].

[34] ujjwalkarn, «An Intuitive Explanation of Convolutional Neural Networks,» the data science blog, 11 Agost 2016. [En línea]. Available: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/. [Último acceso: 28 Maig 2012].

[35] Javier, «Faster R-CNN: Down the rabbit hole of modern object detection,» tryo labs, [En línea]. Available: https://tryolabs.com/blog/2018/01/18/faster-r-cnn-down-the-rabbit-hole-of-modern-object-detection/. [Último acceso: 30 Maig 2021].

[36] L. Weng, «Object Detection for Dummies Part 3: R-CNN Family,» Lil'Log, 31 Desembre 2017. [En línea]. Available: https://lilianweng.github.io/lil-log/2017/12/31/object-recognition-for-dummies-part-3.html. [Último acceso: 30 Maig 2021].

[37] G. G. P. D. R. G. Kaiming He, «Mask R-CNN,» 24 Gener 2018. [En línea]. Available: https://arxiv.org/pdf/1703.06870.pdf. [Último acceso: 1 Juny 2021].

[38] D. Godoy, «Understanding binary cross-entropy / log loss: a visual explanation,» Towards Data Science, 21 Novembre 2018. [En línea]. Available: https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a. [Último acceso: 31 Maig 2021].

[39] COCO, «COCO Home,» [En línea]. Available: https://cocodataset.org/#home. [Último acceso: 2 Juny 2021].

[40] L. Jiao, «A Survey on the New Generation of Deep Learning in Image Processing,» 2019. [En línea]. Available: https://www.researchgate.net/publication/337746202_A_Survey_on_the_New_Generation_of_Deep_Learning_in_Image_Processing. [Último acceso: 30 Maig 2021].

[41] L. Songxiang, «An Introduction to TensorFlow,» 24 Març 2017. [En línea]. Available: https://becominghuman.ai/an-introduction-to-tensorflow-f4f31e3ea1c0. [Último acceso: 1 Juny 2021].

[42] Keras, «About Keras,» [En línea]. Available: https://keras.io/about/. [Último acceso: 1 Juny 2021].

[43] L. A. T. V. Eli Stevens, «Deep Learning with PyTorch,» 4 Agost 2020. [En línea]. Available: https://pytorch.org/assets/deep-learning/Deep-Learning-with-PyTorch.pdf. [Último acceso: 3 Abril 2021].

[44] Scikit-image, «Skeletonize,» [En línea]. Available: https://scikit-image.org/docs/dev/auto_examples/edges/plot_skeleton.html. [Último acceso: 25 Març 2021].

[45] Agencia Tributaria, «Tabla de coeficientes de amortización lineal,» [En línea]. Available: https://www.agenciatributaria.es/AEAT.internet/Inicio/_Segmentos_/Empresas_y_profesionales/Empresas/Impuesto_sobre_Sociedades/Periodos_impositivos_a_partir_de_1_1_2015/Base_imponible/Amortizacion/Tabla_de_coeficientes_de_amortizacion_lineal_.shtml. [Último acceso: 6 13 2021].

[46] R. Calvo, «Mesura i calibratge d'espècies, Treball de final de Grau, UPC,» Escola Tècnica Superior d'Enginyeria Industrial de Barcelona, 2021.