Treball de Fi de Màster

Màster en Enginyeria en Tecnologies Industrials

Identificació automatitzada de gambes amb xarxes neuronals de convolució

MEMÒRIA

Autor: Llorenç Piera Grau
Director: Vicenç Parisi Baradad

Convocatòria: Septembre 2020

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

Resum

Aquest projecte tracta l'automatització dels preus de gambes en funció de la seva mida. Les llotges catalanes ofereixen un servei de venda online que necessita un procés d'automatització per reduir costos i augmentar la confiança dels compradors. Actualment la fixació del preu de les gambes es fa de forma manual i suposa temps i recursos humans. Per altra banda els compradors tenen poca confiança en aquests preus ja que no són capaços de veure si el preu de les gambes es correspon amb la seva mida.

El projecte persegueix l'objectiu de crear un model de *machine learning* que sigui capaç d'identificar el nombre de gambes d'una imatge. Coneixent el nombre de gambes i el pes d'aquestes, es podrà automatitzar la fixació dels preus de les gambes. Això permetrà reduir costos d'estructura, serà un criteri que generarà confiança en els potencials clients i incentivarà la compra.

Amb la finalitat d'aconseguir els objectius s'ha dut a terme un estudi dels recursos de *machine learning* disponibles avui dia i s'ha documentat una detallada base teòrica. Aquest estudi ha servit per conèixer les possibles vies de desenvolupament del projecte i pot servir de guia per introduir-se al món del *machine learning* per a futurs investigadors.

Després d'aquest llarg procés d'estudi s'ha pogut identificar una via de desenvolupament que podria permetre assolir l'objectiu. S'ha decidit entrenar un algoritme de convolució anomenat Faster-RCNN. Per optimitzar l'ús de les imatges disponibles s'ha decidit utilitzar un algoritme pre-entrenat amb la base de dades COCO.

El resultat de la experimentació ha estat un model capaç d'identificar les gambes amb la precisió necessària per complir amb els requisits de les llotges catalanes. El desenvolupament d'aquest projecte obre la porta a una oportunitat per materialitzar el projecte a través d'una instal·lació del model a les línies d'entrada del peix de les llotges.

Sumari

1. Prefaci

1.1. Origen del projecte

L'origen d'aquest projecte ve de les llotges de peix de Catalunya. Aquestes s'estan iniciant a la venta online i s'han trobat amb un problema de confiança entre venedor i comprador. Els clients d'aquest producte online no se'n refien de que el producte que rebin correspongui amb el preu que se'ls cobra. Per exemple, quan es compra gamba a la peixateria és possible veure la mida de la gamba i valorar si el preu que es cobra és just o no. El problema de fer aquesta compra online és que no es pot fer una valoració de si el preu de venta s'ajusta a la mida de la gamba o no. És per això que les llotges han d'establir un criteri objectiu de fixació del preu que generi confiança al comprador. Aquesta mesura és el nombre de gambes per quilogram de producte. Com menys unitats per quilogram, més grans les gambes i més alt el preu. Per tant, el repte està en crear un mètode automàtic i objectiu d'establiment del preu.

1.2. Motivació

L'interès personal per aquest projecte ve donat tant pel fet de poder desenvolupar una tasca relacionada amb una empresa externa a la universitat com per la temàtica del projecte en si. Em trobo en l'últim any d'estudis i a aquestes alçades m'hagués agradat fer practiques per tenir un primer contacte amb el món laboral. He volgut aprofitar aquest projecte per col·laborar amb alguna entitat externa degut a que practico un esport que em requereix molta dedicació i no m'ha permès disposar de temps per a les pràctiques a l'empresa.

La temàtica del projecte també em sembla altament atractiva. Observo que gran part del mon tecnològic està centrat en la creació de màquines que prenguin decisions intel·ligents. Al llarg de la carrera no he tractat aquest tema i he volgut aprofitar aquesta oportunitat per fer-ho.

1.3. Requeriments previs

El principal requeriment previ a l'inici del projecte ha estat la formació personal en el camp de la visió per computador. Abans de l'inici del projecte desconeixia l'àmbit en el que es desenvolupa la investigació i ha calgut moltes hores de lectura i el seguiment d'un curs *online* per assolir els coneixements bàsics necessaris. Aquesta formació ha suposat molt

temps però ha tingut resultats molt positius.

A part de la formació personal, també calia recol·lectar imatges per fer l'entrenament de les xarxes neuronals. Aquest requeriment ha estat desenvolupat per el professor Vicenç Parisi Baradad, persona de contacte amb les llotges.

2. Objectiu i abast del projecte

El projecte te la finalitat d'ajudar a les llotges de Catalunya a automatitzar el procés de *pricing* d'un dels seus productes, les gambes. Les llotges han començat a vendre el peix online i tenen problemes amb els productes que tenen un preu variable en funció de la seva mida, com és el cas de les gambes. Han experimentat un rebuig cap aquest tipus de productes i es creu que és degut a que el client no te suficient confiança en la plataforma *online* i el preu que s'assigna. Sabent això, les llotges volen establir un criteri de preus objectius que generi més confiança en el client i incentivi la compra. Aquest nou criteri es basa en 2 variables: el nombre de gambes de la capsa i el pes de la capsa. D'aquesta manera es pot establir un pes mitjà de les gambes de la capsa i fixar un preu amb un criteri objectiu. En línia amb aquest objectiu general s'ha fixat un objectiu per al projecte.

L'objectiu del projecte és la creació d'un model de *machine learning* capaç de reconèixer i comptar el nombre de gambes presents a una imatge. Aquest objectiu no és l'únic requisit que es necessita per l'establiment dels preus, però si que és la tasca que aporta més valor.

L'assoliment o no d'aquest objectiu pot variar molt en funció de la imatge en estudi. Variables com la quantitat de gambes, la disposició de les gambes, la qualitat de la imatge i el fons de la imatge poden fer variar molt el resultat. Per definir quin és el tipus d'imatge que el model ha de ser capaç d'analitzar s'han escollit les imatges proporcionades per les pròpies llotges.

3. Base teòrica

3.1. IA, Machine Learning i Deep Learning

Intel·ligència artificial, *machine learning* i *deep learning* són termes molt utilitzats en el context de digitalització actual. Cadascun, però, te un significat diferent.

3.1.1. Intel·ligència artificial

Intel·ligència artificial fa referència a la capacitat de les màquines per prendre decisions amb una intel·ligència semblant a la humana. És un terme complicat de determinar ja que és complex saber si un ordinador realment es comporta de manera intel·ligent. John McCarthy diu que "és fer que una màquina es comporti d'una manera que seria considerada intel·ligent en un ésser humà". És el terme més general dels tres.

3.1.2. Machine learning

Machine learning és un camp de la intel·ligència artificial enfocat a l'estudi d'algoritmes que milloren de forma automàtica a través de l'experiència. Els algoritmes de *machine learning* construeixen models matemàtics basats en les dades d'entrenament per fer prediccions o decisions sense estar específicament programats per fer-ho. Hi ha múltiples tipus de *machine learning* en funció del tipus d'informació que reben com a *input* i *output*, i en funció del tipus de problema o tasca per la que estan dissenyats. Els tres tipus de *machine learning* més reconeguts són: *supervised learning*, *unsupervised learning* i *reinforncement learning.*

Supervised learning obté els models a través d'un procés on s'introdueixen dades com a *input* de l'algoritme, es fa una predicció i es dona *feedback* a l'algoritme sobre si ha fet una bona predicció o no. Per tant, cal disposar de dades etiquetades per entrenar l'algoritme. Un exemple de dada etiquetada és una imatge d'un gat amb una etiqueta que la classifica com a "gat". Quan un algoritme està entrenat, s'obté un model que és capaç d'etiquetar dades que mai ha vist abans. Per exemple, si hem entrenat l'algoritme a reconèixer si l'animal d'una imatge és un gat o no, el model serà capaç d'etiquetar una imatge que mai ha vist en funció de si conté un gat o no. Altres aplicacions de *supervised learning* són: la identificació del correu *spam*, el reconeixement facial o els criteris de selecció d'anuncis online.

Unsupervised learning es diferencia del *supervised learning* perquè no requereix el *feedback*. En canvi, aquests algoritmes s'alimenten de dades i se'ls dona eines per

entendre les propietats d'aquestes dades. Aquest camp del *machine learning* és molt interessant per poder utilitzar tota aquella informació no etiquetada i treure'n un rendiment. Alguns exemples d'aplicació d'aquest tipus d'aprenentatge són: els sistemes de recomanació de contingut que utilitzen Netflix o Youtube o la classificació de clients en funció dels seus hàbits de compra per detallar la segmentació del mercat.

Reinforcement learning és un tipus d'aprenentatge basat en la interacció entre un element i un entorn. La manera en la que l'element i l'entorn interaccionen a través d'accions preses per l'element es pot classificar de forma positiva o negativa en funció dels interessos pertinents. D'aquest forma s'aconsegueix promoure les accions positives i evitar les negatives. Un exemple d'aquest tipus d'aprenentatge són els robots de línies de producció. Per programar les tasques d'aquests robots de forma econòmica i segura s'utilitza el *reinforcement learning.*[1]

3.1.3. Deep learning

L'altre terme que està revolucionant la tecnologia de la informació és el *deep learning*. Aquest concepte no és més que l'aplicació del *machine learning* a través de xarxes amb molts nivells, profundes. Fa anys que aquest concepte està present, però no ha estat fins recentment que ha començat a tenir un impacte important. El fet que ha marcat l'inici del *deep learning* ha estat la digitalització. La digitalització, al crear grans quantitats de dades, ha permès que les xarxes neuronals més profundes tinguin un rendiment molt superior a les de menor nombre de nivells. El rendiment d'un model depèn de la quantitat de dades que l'han entrenat i de la mida de la xarxa neuronal. Abans de la digitalització el rendiment dels models es veia limitat per la quantitat de dades que es disposava, no per la mida de les xarxes neuronals. Actualment es generen tantes dades que és possibles aprofitar aquestes xarxes neuronals d'alta profunditat. Aquest és el motiu del recent esclat del *deep* learning.

3.2. Tipus d'input

Els inputs que s'introdueixen als algoritmes poden ser molt diversos, però es poden agrupar en si es tracta d'informació estructurada o no estructurada.

La informació estructurada és amb la que estem més acostumats a treballar i analitzar. Exemples de dades estructurades són noms, dates, adreces, números de targetes de crèdit, geolocalitzacions... Són dades altament organitzades i interpretables. A dia d'avui l'anàlisi i manipulació d'aquest tipus de dades és relativament fàcil i ràpid en comparació amb les dades no estructurades. Un dels llenguatges de programació per utilitzar aquest tipus de dades és el SQL (Structured Query Language), desenvolupat per IBM a la dècada

de 1970. A simple vista és un tipus d'informació molt difícil d'utilitzar, però a traves d'eines d'anàlisi es poden descobrir patrons i tendències valuoses per aplicacions de *machine learning.*

Les dades no estructurades s'acostumen a qualificar com qualitatives i són més difícils d'analitzar i processar amb eines i mètodes convencionals. Exemples de dades no estructurades són text, vídeo, àudio, imatges i activitat de xarxes socials, entre d'altres. Aquest tipus de dades són difícils d'organitzar i manipular. Més del 80% de les dades generades es considera no estructurada. Trobar informació útil a partir de dades no estructurades és una tasca que requereix un anàlisi avançat i un alt nivell de coneixements tècnics. [2]

Els humans tenim una capacitat d'anàlisi de dades no estructurades molt alta, però tenim dificultats alhora de processar grans quantitats de dades estructurades. Per contra, els ordinadors fa anys que tenen eines molt avançades pel tractament de dades estructurades però fins recentment no han començat a analitzar dades no estructurades. Gràcies a les xarxes neuronals els ordinadors són cada cop millors en processar informació no estructurada.

La intel·ligència artificial ha creat un gran impacte a la societat perquè hem vist que els ordinadors també són capaços d'interpretar dades no estructurades. Pot semblar molt sorprenent que un ordinador detecti un gat a una fotografia, és una capacitat que fins ara només atribuíem a humans i altres éssers vius. Tot i això, la intel·ligència artificial també està creant grans avenços en el tractament i anàlisi de dades estructurades. El negoci que ha sorgit de personalització d'anuncis online és un clar exemple de l'impacte econòmic de la intel·ligència artificial a través de dades estructurades.

3.3. Algoritme i model

Un altre parella de termes fàcilment confosos són l'algoritme i el model. L'algoritme és el procediment a través del qual s'assoleix l'aprenentatge sobre una certa informació. Per assolir l'aprenentatge, l'algoritme fa un procés iteratiu de prova i error en el que poc a poc es va ajustant millor a les dades d'estudi.

El model, en canvi, representa tots aquests coneixements apresos durant l'entrenament juntament amb un algoritme de predicció. A diferència de l'algoritme d'aprenentatge, l'algoritme de predicció no varia ni aprèn, és un procés constant que conté la informació del model per fer les prediccions. Per tant, podem dir que un model està composat per un conjunt de dades fixes i un algoritme de predicció.

El gran tret diferencial és que l'algoritme és un procés d'aprenentatge i el model és un conjunt de dades per fer prediccions. El model és el nostre objectiu i l'algoritme és el mètode d'aconseguir-lo.[3]

3.4. Neurones

A l'estudi que ens ocupa utilitzarem *supervised* learning. El mètode iteratiu amb el que s'entrena un algoritme d'aquest tipus pot ser de regressió o de classificació en funció del seu *output*. Als models de regressió els *outputs* són continus, en canvi als models de classificació són discrets. La visió per computador per detectar gambes és un model de classificació que tindrà com a output un "1" si la imatge és una gamba i un "0" si no ho és, per tant es tracta d'un model de classificació discret.

Tant pels models de regressió com pels de classificació existeixen diferents tipus d'algoritmes. Pels models de regressió existeix la regressió lineal, l'arbre de decisió, el *random forest* o les xarxes neuronals. Pels models de classificació podem utilitzar algoritmes de *logistic regression*, *support vector machine* i també arbres de decisió, *random forest* i xarxes neuronals. [4]

En el nostre cas utilitzarem les xarxes neuronals per adreçar-nos a l'objectiu de classificació d'imatges.

Una xarxa neuronal és una estructura de neurones artificials amb connexions entre elles. Una neurona és una funció que agafa certs valors com a *inputs* (x_1, x_2, x_3... x_n,) i produeix un output (y). Hi ha diferents tipus de neurones que utilitzen diferents funcions per calcular l'output. La neurona més senzilla, el perceptró, multiplica cada input binari per un pes (w_1, w_2, w_3... w_n) i suma tots els factors. Si el resultat d'aquesta suma està per sobre d'un llindar, l'output és 1, si està per sota del llindar, l'output és 0. Si escrivim aquesta funció prenent w com un vector dels pesos, x com un vector dels inputs i expressant el llindar a l'altre costat de l'equació, ens queda la següent expressió:

$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

Eq 1

Hem passat a representar el llindar amb una "b" de *bias.* Aquest és el terme més utilitzat a la comunitat de *machine learning*.

Un altre tipus de neurona és la de tipus *sigmoid.* Una de les diferències respecte al perceptró és que els inputs no cal que siguin binaris, poden ser valors entre 0 i 1. Al igual que al perceptró, es multipliquen per uns pesos i se'ls aplica un *bias*, però l'output no és un valor binari. En comptes de retornar un 0 o 1, s'aplica una funció *sigmoid* (σ):

$$\sigma(z) \equiv \frac{1}{1+e^{-z}}.$$

Eq 2

Aquesta funció retorna un valor entre 0 i 1:

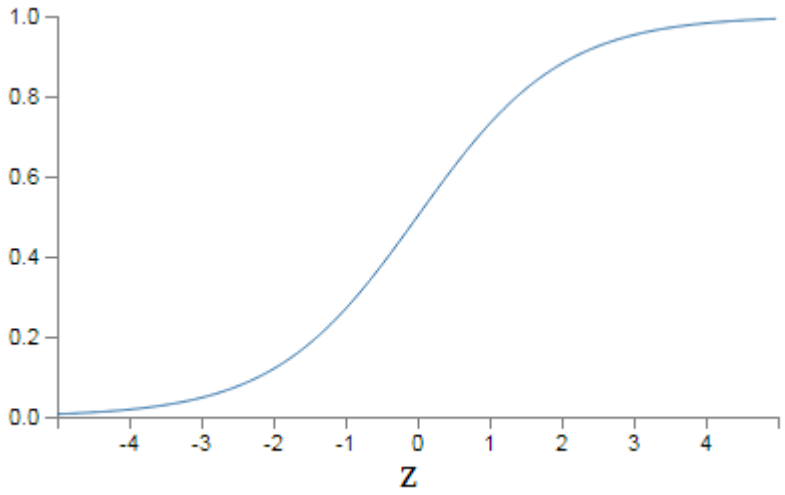


Fig 1 Funció sigmoidal

La utilitat d'utilitzar una funció semblant a la funció graó però suavitzada és que petits canvis en els pesos produeixen petits canvis a l'output. Això és un benefici enfront els perceptrons alhora d'entrenar algoritmes.

Aquests dos tipus de neurones són les més senzilles però no les úniques. N'hi ha amb altres funcions d'activació diferents a la *sigmoid* i amb altres característiques més complexes.[5]

3.5. Arquitectura de les xarxes neuronals

El conjunt d'aquestes neurones forma xarxes neuronals:

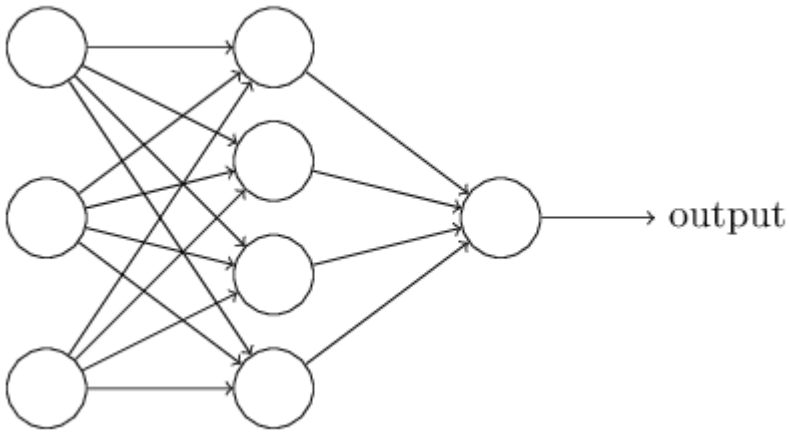


Fig 2 Arquitectura neuronal

El nivell més a l'esquerra de la xarxa s'anomena nivell input i les neurones s'anomenen neurones input. El nivell més a la dreta és el nivell output, que conté les neurones output, en aquest cas només una. Els nivells intermitjos són les *hidden layers,* en aquest cas només un.

El disseny dels nivells input i output de la xarxa neuronal és molt directe si es coneix bé el problema. Posem el cas que es vol determinar si un número manuscrit en una imatge de 64 x 64 píxels és un 9 o no. L'input seria de dimensió 64 x 64 = 4096 neurones. L'output, per altra banda, estaria format per una sola neurona que retornaría un 1 o un 0 en funció de si la imatge conte un 9 o no.

A diferència de l'input i l'output, les *hidden layers* no tenen un disseny directe. En funció dels objectius de cada tasca interessa dissenyar les *hidden layer* d'una manera o altre. Per exemple, si volem un model molt precís per un problema molt complex seria recomanable disposar de múltiples *hidden layers.* Si alhora volem que el temps d'entrenament estigui per sota d'un cert temps ens trobarem en suggerències contradictòries pel mateix problema. És per això que hi ha molta recerca sobre com afecten les diferents característiques de les *hidden layers* al rendiment del models.

Fins ara hem vist xarxes neuronals *feedforward,* on l'output d'un nivell és l'input del següent. Existeixen altres xarxes neuronals on l'input d'una neurona depèn de l'output de la mateixa neurona. Aquests models s'anomenen *recurrent neural networks* i s'utilitzen per tractar informació seqüenciada en el temps. La traducció i la transcripció són dos exemples on l'aplicació de *recurrent neural networks* és molt útil ja que el significat de les paraules varia en funció de les paraules predecessores.[5]

Un altre tipus de xarxes neuronals són les xarxes de convolució, que s'utilitzen en el

tractament d'imatges. Quant l'input d'una xarxa neuronal és una imatge de 1000 x 1000 píxels vol dir que el nivell input és de 3 milions de nodes, ja que per cada píxel tenim informació dels 3 colors RGB (1000 x 1000 x 3 = 3M). El tractament d'aquest tipus d'input amb estructures totalment connectades on el node d'un nivell està connectat a tots els nodes del següent nivell, generaria estructures amb una complexitat molt alta. Per aquest motiu van sorgir les xarxes de convolució. En comptes d'analitzar tots els píxels a la vegada es fa un estudi per regions de la imatge, fet que permet reduir la complexitat de les xarxes.

Tipus de xarxes neuronals en funció de la seva aplicació:

Input	Output	Aplicació	Tipus de Xarxa
Característiques d'una casa	Preu	Real State	Feedforward
Anunci, Perfil Usuari	Click a l'anunci o no (0/1)	Anuncis Online	Feedforward
Imatge	Objecte (1....1000)	Classificació d'imatges	Convolució
Anglès	Xinés	Traducció	Recurrent
Àudio	Text	Transcripció	Recurrent
Imatge, radar	Posició d'altres cotxes	Conducció autònoma	Híbrid

Taula 1

3.6. Conceptes bàsics de programació de xarxes neuronals

3.6.1. Classificació binària

Per entendre millor què és la classificació binària continuarem amb l'exemple de classificació d'imatges de gats. En aquest cas l'input són imatges. Una imatge es representa a l'ordinador com 3 matrius amb tantes caselles com píxels hi ha a la imatge. Cada matriu correspon a un color RGB. L'input es caracteritza amb la lletra "x", que correspon a un vector amb tots els valors dels píxels d'una imatge posats en columna.

Habitualment tenim múltiples exemples d'entrenament que introduirem com a input x. Per

això és comú ajuntar totes les columnes en una mateixa matriu, cada columna correspon a una imatge. D'aquesta manera aconseguim que l'input "x" agrupi diverses imatges alhora. Per notar que l'input es tracta d'una agrupació d'exemples utilitzarem la "X".

L'output "y" també s'agrupa en fila, on cada columna correspon a un valor amb valor 0 o 1 en funció de si la imatge d'aquella columna és un gat o no. Al ser una agrupació de outputs també utilitzarem la notació "Y". Amb aquesta notació queda molt ordenat tant l'input com l'output.

Durant la fase d'entrenament l'algoritme rebrà imatges i farà una predicció de si creu que aquella imatge conté un gat o no. Aquesta fase es diu *forward propagation.* Un cop feta, la predicció es compara amb l'output real d'aquella imatge i es calcula un error de predicció. Un cop calculat l'error s'inicia la fase de *backward propagation*, que busca corregir els pesos "w" i el bias "b" de les diferents per que millorin les seves prediccions.

El nom de classificació binària ve determinat pel fet que l'output només pot prendre dos valors.

3.6.2. Model de regressió logarítmica

Una de les opcions per fer el *forward* propagation és el model de regressió logarítmica. A partir d'una imatge, l'algoritme fa una predicció de si aquella imatge conté un gat o no. Per fer-ho utilitza neurones de tipus *sigmoid*.

Donada una X, volem saber predir l'output. La predicció de l'output s'expressa amb la lletra Ŷ. ŷ és la probabilitat de que y=1 per una imatge x. Per exemple, ŷ és la probabilitat de que una imatge contingui un gat. Per tant, veiem que ŷ no és una variable discreta, sinó que pren un valor entre 0 i 1 seguint la distribució *sigmoid*

Cal recordar que intervenen els paràmetres de pes i *bias* (w i b):

$$\hat{y} = \sigma(w^t \cdot x + b)$$

Eq 3

$$\sigma = \frac{1}{1 + e^{-z}}$$

Eq 4

3.6.3. Funcions *Loss* i *Cost* del model de regressió logarítmica

Una vegada feta la predicció, es calcula la funció *Loss,* que calcula l'error entre l'output real i la predicció. N'hi ha de diverses, però un exemple de funció *Loss* és:

$$Loss = L = \frac{1}{2}(\hat{y}(w,b) - y)^2$$

Eq 5

La funció *Loss* calcula l'error d'un sol exemple. Per fer un còmput global de l'error sobre tots els exemples s'utilitza la funció *Cost*:

$$Cost = J = \frac{1}{m}\sum_{i=1}^{m} Loss(w,b)$$

Eq 6

3.6.4. Gradient descent

La funció *Cost* depèn de dels paràmetres w i b. L'objectiu és trobar els valors de w i b que minimitzen la funció *Cost.* Un mètode iteratiu per trobar aquests valors és el *Gradient Descent.*

El mètode del *Gradient Descent* calcula els pendents en funció de cada una de les variables i actualitza aquella variable amb un valor que redueix el valor de la funció *Cost*. És a dir, es calculen l'impacte de cada variable sobre la funció *Cost* i es busca reduir aquest impacte iteració a iteració. La forma de calcular l'impacte de cada paràmetre sobre la funció J és a través de derivades. D'aquesta manera s'aconsegueix aproximar-se al mínim de la funció J(w,b). Cada iteració s'apropa al punt més baix de la funció, s'apropa a l'error més petit.

El paràmetre que estableix quant es canvien els valors dels paràmetres w i b és el *learning rate.* Un cop calculat el pendent en funció de cada variable s'ha de canviar el valor de la variable en la direcció del pendent negatiu, però de quina dimensió és aquest canvi? Aquesta mida del pas de descens és la que marca el *learning rate.*

Hi ha diferents opcions respecte a la freqüència d'actualització dels paràmetres. La opció

més intuïtiva, tot i que no te perquè ser la millor, és que s'actualitzin després de cada iteració. Més endavant veurem altres opcions.

3.6.5. Inicialització de paràmetres

Un dels aspectes a tenir en compte alhora d'entrenar un algoritme és el valor d'inicialització dels paràmetres w i b. A les xarxes neuronals no és correcte inicialitzar aquests valors en 0 perquè totes les neurones tindrien el mateix pes sobre la funció *Cost* i després de cada iteració variarien de la mateixa forma i serien sempre iguals entre ells. Aquest fet s'anomena simetria en els paràmetres.

Una forma d'inicialitzar els paràmetres és de forma aleatòria, però cal anar en compte en que aquests valors aleatoris no prenguin valors massa grans o massa petits en funció de la seva funció d'activació. Per funcions com la *sigmoid* o la *tanh* cal tenir valors propers a 0 per evitar caure en zones de la funció on el pendent és molt petit. Si això passés causaria un ritme d'aprenentatge molt lent degut a que cada iteració canviaria molt poc el valor dels paràmetres. Hi ha altres funcions com la ReLU on això no suposaria un problema.

3.6.6. *Hyperparameters*:

Ja hem introduït els dos paràmetres w i b que varien durant l'entrenament per optimitzar el model. Hi ha altres variables amb influència directa en el rendiment del model que es poden triar per tenir un impacte positiu en el model. Aquests paràmetres s'escullen abans d'iniciar l'entrenament i s'anomenen *hyperparameters* per diferenciar-los dels primers.

Alguns exemples dels *hyperparameters* més bàsics són: el *learning rate*, el nombre d'iteracions d'entrenament, el nombre de *hidden layers*, el nombre de *hidden nodes*, el *momentum*, la mida del *batch*, els paràmetres de regularització...

Ajustar els *hyperparameters* pot ajudar molt al comportament de la xarxa neuronal. Una manera de triar-los és estudiant la funció *Cost* en funció dels diferents *hyperparameters* i escollint aquella combinació que minimitzi la funció.

És difícil saber d'avant mà què és el que funcionarà millor. Per això és important fer un procés empíric per decidir què aplicar en funció del seu comportament.

3.7. Aspectes pràctics de les xarxes neuronals

3.7.1. *Training validation* i *test sets*:

Per entrenar un algoritme cal una base de dades. Una part d'aquestes dades serveixen per

entrenar l'algoritme i una altra per avaluar el rendiment de l'algoritme. No podem avaluar el rendiment de l'algoritme amb les mateixes dades que utilitzem per l'entrenament ja que és possible que el model estigui especialitzat només en els exemples concrets de l'entrenament. Si això passés, tindríem un rendiment teòric molt alt però a l'aplicació final veuríem uns resultats pobres. Per evitar que això passi es divideix la base de dades en 2 o 3 subgrups. Aquests altres subgrups ens serviran per avaluar de forma objectiva el model.

Una possibilitat és dividir la base de dades en 3 subgrups. El primer és el *training set,* que conté aquelles dades que s'utilitzaren per fer l'entrenament. El segon és el *validation set* o també conegut com *dev set.* Aquest segon subgrup s'utilitza per estudiar el rendiment del model en funció de les diferents configuracions de *hyperparameters.* És a dir, serveix per estudiar la configuració idònia de *hyperparameters.* A més a més també serveix per assegurar que no hi ha una sobre especialització del model sobre les dades del *training set*. El tercer és el *test set,* que serveix per assegurar que la configuració de *hyperparameters* escollida no ha resultat en una especialització del model en el *dev set.* El *dev* i *test sets* han de ser iguals, que persegueixin el mateix objectiu. Si no fos així, s'hauria preparat al model per rendir en un base de dades que no representa el mateix que el *train set*. Separar aquests dos sets i avaluar-los per separat és una bona manera d'assegurar-se que s'està treballant pel mateix objectiu. Per tant, cal triar un *dev* i *test set* de la mateixa distribució i que representi les dades amb les que s'espera tenir un bon rendiment en el futur. Que s'assembli al que després ens trobarà a la realitat

La divisió de les dades en aquests grups depèn de la mida de la base de dades. Una manera de dividir és de 60% pel training set, 20% pel dev set i 20% pel test set. Aquesta distribució és correcte si no tenim una gran quantitat de dades i per això ha estat una pràctica comú des de l'inici del machine learnin. Actualment, però és possible que tinguem bases de dades de 1M d'exemples o més. En aquests casos no cal dedicar un percentatge tant elevat al dev set i al test set. Un 1% en cadascun d'ells ja suposa 10.000 exemples de cada.

Una altra opció és dividir les dades en dos grups: el training set i el dev set. El training set serveix per entrenar i el dev serveix per ajustar els paràmetres de la millor manera possible. Aquesta és una pràctica que ha portat confusió perquè una part de la comunitat anomena al dev set, training set. Això no és del tot correcte ja que si ajustes els paràmetres i segueixes el procés iteratiu després d'haver fet l'entrenament, el que estàs fent és utilitzar un dev set. No és recomanable no tenir un test set perquè no tens el set que t'assegura una avaluació no esbiaixada del model. Existeix el risc de que la configuració dels hyperparameters estigui especialitzada només pel dev set i que el rendiment baixi al implementar el model.

Idealment les dades del training, dev i test sets haurien de ser el més semblants possibles a

les de l'aplicació final. Del contrari, desvirtuant l'entrenament o l'avaluació. Anomenem les dades de l'aplicació final com dades de referència, ja que són les que volem saber analitzar el millor possible. Per algunes projectes no és possibles trobar suficients dades amb la mateixa distribució que l'aplicació final i cal recórrer a dades lleugerament diferents. En aquests casos cal triar com distribuir les dades de referència en els diferents subgrups.

En aquest projecte ens hem trobat amb poques imatges semblants a les que es farien servir a les llotges. Per tant, la distribució de les imatges en els diferents subgrups ha estat un tema clau. Al disposar de tant poques imatges de referència s'ha decidit prescindir del test set. Per altra banda s'ha assegurat que el dev set sigui el més fiable possible, conformant-lo completament d'imatges de referència. S'ha prioritzat el fet de tenir una avaluació fiable enfront la màxima qualitat de l'entrenament.

3.7.2. *Bias* i *variance*

Un cop fet l'entrenament d'un algoritme cal avaluar el seu rendiment. Dos conceptes que ens ajuden a comprendre el comportament dels models són el *bias* i la *variance.*

El bias és l'error del *training set*, quant lluny està de l'error 0. Ens dona una idea de quina es la millora potencial del nostre entrenament. A vegades és impossible arribar a un error 0 i cal definir un error objectiu major que 0. La diferència entre l'error objectiu i l'error del *training set* és el *avoidable bias.* Cal ser conscients de quin és el mínim error que cal assumir. En funció de l'aplicació haurem d'acceptar un error més gran o més petit, però aquest error serà el mínim error que podem aconseguir i no voldrem rebaixar-lo. Aquest error mínim s'anomena *Bayes error*. Nosaltres assumirem que el *Bayes error* és igual a l'error humà, per poder-lo fixar de forma més fàcil. En realitat l'error humà sempre és més gran que el *Bayes error*, però nosaltres assumirem que és el mateix. Per tant, *l'avoidable bias* serà la diferència entre l'error humà i el *training error* i el nostre objectiu serà minimitzar-lo.

La definició anterior suposa que un model mai podrà actuar millor que un humà. Tot i que no és del tot cert, per al cas que ens ocupa és una suposició assumible. Els algoritmes podran actuar millor que els humans en tasques de percepció no natural. Es a dir, dades estructurades. Els humans no som capaços d'analitzar dades estructurades tal i com ho fa un algoritme. En canvi, en tasques de percepció natural com reconeixement de veu o reconeixement d'imatges és més difícil sobrepassar el nivell humà. Tot i això, actualment ja hi ha algunes tasques de percepció natural que actuen millor que els humans.

L'altre valor que ens ajuda a caracteritzar el model és la diferència entre el *training error* i el *dev error,* la *variance.* La *variance* és l'indicador que ens informa de si l'entrenament del model s'ha especialitzat només sobre el *training set* o no. L'especialització sobre el *training*

set rep el nom de *overfitting* i és negatiu pel model, interessa evitar-lo.

En funció dels valors del *bias* i la *variance* actuarem sobre els paràmetres de l'algoritme de diferent manera.

La taula següent mostra alguns exemples de *bias i* variance:

Human error	0.5%	0.5%	0.5%	0.5%
Training error	1%	15%	15%	0.5%
Dev error	11%	16%	30%	1%
Observacions:	Alta *variance*	Alt *bias*	Alt *bias* Alta *variance*	Baix *bias* Baixa *variance*

Taula 2

És prioritari donar dades al model per entrenar-lo amb el màxim d'imatges possible. La conseqüència és que les dades d'entrenament acaben tenint una distribució diferent a la de l'aplicació final. Això passa perquè no disposem de suficients dades amb exactament la mateixa distribució que l'aplicació. No te perquè ser un problema el fet d'utilitzar dades per l'entrenament que no tinguin la mateixa distribució que el *dev* i *test sets*. Tot i això, cal tenir en compte que la *variance* es veurà afectada. És possible que l'error del *dev set* vingui causat per aquesta diferència en la distribució de les dades i no pel sobre-entrenament del *training set.*

Per assegurar que l'error es degut a la diferència entre distribucions, es crea un nou subgrup anomenat *training-dev set*. Aquestes dades tenen la mateixa distribució que les dades d'entrenament i serveixen per comprovar si la *variance* és deguda al sobre-entrenament de les dades o si és deguda a la diferent distribució de les dades.

En aquests casos, per estudiar el comportament del model observarem *Bayes error*, el *training set error*, el *traning-dev set error*, el *dev error* i el *test error*. Si calculem la diferència entre els diferents valors obtindrem, *avoidable bias*, *variance*, error per diferencia en la distribució i *variance* sobre el *dev set*.

3.7.3. Millorar el rendiment del model

Un dels principals objectius del procés iteratiu d'entrenament d'un algoritme és reduir al

mínim *l'avoidable bias* i la *variance.* Hi ha diferents mesures que es poden prendre per millorar aquest indicadors.

Una opció per reduir *l'avoidable bias* és incrementar el temps d'entrenament. És possible que l'algoritme encara no hagi convergit i es pugui millorar amb només un increment del temps d'entrenament. Una altra mesura és el canvi de xarxa neuronal capa a una xarxa més complexa. El fet d'augmentar la complexitat de la xarxa pot ajudar a interpretar les dades millor. Altres opcions per millorar l'*avoidable bias* són canviar els *hyperparameters o* canviar d'arquitectura de xarxa neuronal,

Una mesura per tractar problemes de *variance* és incloure més dades al *training set*, d'aquesta forma és més difícil que el model resulti en *overfitting.* Una altra opció és utilitzar el mètode de regularització. Per a la *variance* també pot ser beneficiós un canvi de xarxa neuronal o una modificació dels *hyperparameters*.

Si el que tinc són problemes per la diferència en la distribució de les dades entre el *training* i el *dev set,* s'ha d'analitzar en què són diferents les dades d'entrenament i les del *dev*. Un cop identificat, podem intentar modificar les dades d'entrenament per que s'assemblin a les de *dev*. Una de les eines per tenir més dades semblants a les dades del *dev set* és crear-les de forma artificial. Per exemple, si volem entrenar un algoritme de reconeixement de veu per un cotxe, superposar soroll del cotxe sobre gravacions de veu. Aquesta tècnica de sintetitzat funciona, però cal que aquests annexes artificials siguin suficientment grans per que l'algoritme no creï *overfitting* sobre aquestes dades.

3.7.4. Regularització

És una solució per models amb un elevat nivell de *variance* on no és possible aconseguir més dades de. És una mesura que pot perjudicar el *bias,* cal avaluar si la baixada de *variance* compensa l'augment del *bias.*

Quan s'aplica la regularització s'afegeix un terme a l'equació del cost J per simplificar les matrius W, aconseguint valors menors a W. Es simplifica l'efecte dels nivells amagats i per tant simplifica la xarxa neuronal. El fet de que W prengui valors petits fa que treballi en zones més lineals, fet que simplifica la complexitat de la xarxa neuronal i limita la capacitat de acabar en *overfitting*.

3.7.5. Normalització de les dades

La normalització de les dades te l'objectiu de reduir el temps d'entrenament. Consisteix en dues coses. En primer lloc, forçar que la mitjana dels valor sigui 0 en totes les variables. Portem la mitjana a l'origen de coordenades. En segon lloc, normalitzar la variància de les

variables, fer que fer que sigui igual per a totes elles.

S'aplica la normalització perquè quan s'apliqui el mètode de *gradient descent,* el que s'encarrega de portar paràmetres al valor que minimitza la funció cost, l'impacte del *learning rate* sobre cada variable sigui el mateix. Això farà que el temps d'entrenament sigui més petit perquè el pendent de tots els punts de la corba de la funció cost estarà orientat al mínim.

A la figura següent es mostra un exemple gràfic dels beneficis de la normalització en una corba 3D. Tot i que és normal treballar en dimensions molt majors a 3, l'efecte produït és el mateix.

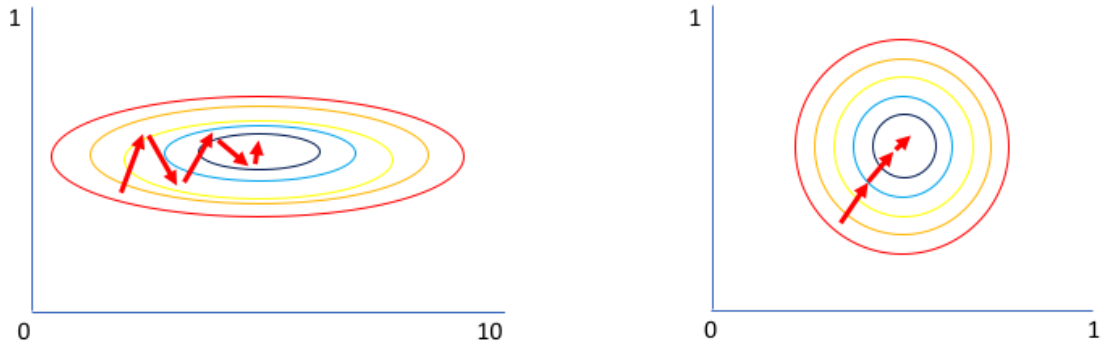


Fig 3 Exemple 3D dels beneficis de la normalització [6]

3.8. Algoritmes d'optimització

3.8.1. *Mini-batch gradient descent*

El mètode del *gradient descent* planteja un procés en el que després de computar el *loss* de cada exemple i posteriorment el *cost* del *training set* complert, es fa un canvi dels paràmetres en la direcció del mínim pendent. Això suposa que cal processar tots els exemples del *training set* per cada canvi dels paràmetres. Tot i que sigui un procés molt fiable en la direcció del pendent, perquè pren la mostra més gran possible, el fa lent.

La proposta de l'algoritme *mini-batch gradient descent* busca reduir el temps entre cada actualització dels paràmetres. Per fer-ho organitza els exemples del *training set* en grups més petits i actualitza els paràmetres després de computar el *cost* de cada grup. Aquests grups s'anomenen *batches.*

Fer un *epoch* és fer un recorregut per tots els exemples del *training set.* Si decidim utilitzar

el *mini-batch gradient descent* amb 1000 *batches,* 1 *epoch* et permet fer 1000 canvis de paràmetre. En canvi, si no tenim *batches*, només seríem capaços de fer 1 canvi de paràmetres.

El resultat d'aplicar el *mini-batch gradient descent* és que reduïm el temps entre cada actualització dels paràmetres però reduïm també la fiabilitat en la direcció d'aquests canvis. Per tant, no és segur que es redueixi el temps global d'entrenament ni que millori la qualitat del model. L'elecció de la mida del *batch* dependrà de cada projecte i caldrà estudiar-la de forma experimental. És interessant que el *batch* sigui un múltiple de 2, perquè es coordina amb els bits de memòria. També cal destacar que el *batch* està limitat per la memòria del CPU/GPU, si és massa gran no serà possible processar cada *batch*.

3.8.2. *Momentum*

Quan implementem el *gradient descent*, pot ser que tinguem grans oscil·lacions en algun dels paràmetres d'entrenament. El *momentum* ens permet ajustar la rapidesa d'aprenentatge en cada direcció de forma independent per reduir les oscil·lacions sense renunciar a la velocitat d'entrenament que aporta un alt *learning rate.*

3.8.3. *Learning rate* variable

Un dels *hyperparameters* que poden influir a la velocitat de l'aprenentatge és el *learning rate*. És interessant poder variar el valor del *learning rate* al llarg de l'entrenament. Quan ens trobem lluny del mínim amb pendents molt negatius ens interessa avançar el més ràpid possible cap al mínim i és assumible la oscil·lació. En canvi, quan ens apropem al mínim ens interessa fer canvis de paràmetre molt petits per arribar el més a prop possible del mínim, oscil·lant a la menor àrea possible.

3.9. Calibrat dels *hyperparameters*

Hi ha múltiples *hyperparameters* que tenen influència en el model i una de les tasques per treure'n el màxim rendiment és el seu calibrat.

3.9.1. Distribució dels valors d'estudi

Quan es vol decidir quina és la millor configuració dels *hyperparameters* pel model d'estudi, cal provar diferents valors de cada *hyperparameter* per entendre el seu impacte sobre el model. Si es vol estudiar l'impacte de diversos *hyperparameters* alhora hi ha diverses opcions alhora de triar els valors d'estudi. Una opció és fer-ho estructuradament en forma de quadrícula i l'altre opció és fer-ho amb valors aleatoris.

La orientació d'aquest projecte envers la distribució dels valors d'estudi és fer-ho de forma aleatòria. Així es creu que s'obtindran molts més valors diferents i es podrà extreure més informació.

Per exemple, si tenim 2 variables i estem disposats a estudiar 25 punts, a l'estructura de quadrícula tindríem 5 valors diferents per cada variable. En canvi amb una orientació aleatòria en tindríem 25.

3.9.2. Escala de valors d'estudi

En ocasions no interessa que l'escala on es reparteixen els valors dels *hyperaparameters* d'estudi sigui una escala lineal. Si es vol repartir el mostreig entre 0,0001 i 1, pot ser més interessant fer-ho en una escala logarítmica que en una escala lineal. En una lineal el 90% de les mostres estarien entre 0,1 i 1. Per tenir el mateix mostreig entre 0,0001 i 0,001 i 0,1-1, es pot fer amb una escala logarítmica. Fer-ho de forma lineal no és bona idea perquè el comportament de mostres que estan properes a 0 és molt diferent per valors molt propers entre ells i interessa tenir moltes mostres. En canvi, la diferència en l'impacte entre valors com el 0,4 i 1 no és tan gran i no interessa tenir tantes mostres. El cas comentat és un exemple de sensibilitat per valors propers a 0, però també es podria donar el cas en valors pròxims a 1.

3.9.3. Pandas vs Caviar

La manera d'afrontar un projecte de *machine leaerning* es pot englobar en dos grans estratègies.

Una opció és treballar amb només un model i insistir en millorar-ho a través de la configuració dels paràmetres. Aquesta estratègia s'anomena Panda en referència als animals que tenen només un fill.

L'altre opció és entrenar diferents models en paral·lel i escollir el que funciona millor. Aquesta segona estratègia s'anomena Caviar, en referència a la reproducció dels peixos.

Aquest projecte tria la estratègia Pandas.

3.9.4. Ortogonalisme

És la capacitat d'una variable de tenir un únic efecte un únic efecte concret sobre l'output. És molt interessant per poder valorar si un canvi sobre un *hyperparameter* és positiu o negatiu pel model. Tot i que és difícil observar un sol efecte dels *hyperparameters,* l'ortogonalisme és un concepte interessant a tenir en compte alhora de triar quins *hyperparameters* millorar.

3.9.5. Mètrica única d'avaluació

És convenient disposar d'un criteri clar de decisió quan volem avaluar si l'efecte del canvi d'un *hyperparameter* ha estat positiu o no. Una de les formes de mirar el rendiment d'un model de reconeixement d'imatge és a través de la *precision* i el *recall. Precision* és el percentatge d'objectes ben identificats sobre el total d'objectes identificats. *Recall* és el percentatge d'objectes ben identificats sobre el total d'objectes a les imatges. Aquests dos valors són un bon indicador de la qualitat del model, però el fet de ser dos valors diferents ens dificulta la presa de decisions sobre els *hyperparameters.* No serà fàcil decidir sobre una configuració que millora el *recall* però empitjora la *precision.*

Per això és recomanable crear una variable que unifiqui les dues variables d'interès. D'aquesta manera només haurem de mirar una mètrica. Un exemple de funció d'unificació és la F1 Score:

$$F1\ Score = \frac{2}{(\frac{1}{precision} + \frac{1}{recall})}$$

Eq 7

3.10. Anàlisi de l'error

3.10.1. Anàlisi qualitatiu

Fins ara ens hem centrat en l'anàlisi quantitatiu per millorar els models. A vegades, però, és convenient fer un anàlisi qualitatiu. Fer un recompte manual de quin tipus d'error hi ha, pot ajudar molt a saber quin tipus de dades està generant problemes.

Analitzar el tipus d'error és interessant per saber què és el que no funciona i poder prendre mesures adequades per aquest tipus de problema. Per exemple, si a un projecte de visió per computador analitzem els errors del *dev set* i detectem que el model no identifica els objectes petits, podrem prendre mesures concretes per aquest tipus d'error. No cal analitzar tots els erros, amb una mostra aleatòria n'és suficient.

3.10.2. Construir a partir de l'error

Intentar dissenyar l'algoritme perfecte en un intent és molt complicat perquè cal predir quines són les dificultats del projecte sense haver-lo provat abans. Per això és recomanable entrenar un primer algoritme senzill i construir a partir dels errors observats en aquesta

primera iteració. Això ens centrarà en quines són les dificultats de l'aplicació i podrem enfocar-nos en els aspectes que seran realment importants.

3.11. Transferència de coneixements

Quan s'inicia un projecte nou de *machine learning* és interessant investigar quins projectes semblants ha desenvolupat la comunitat. Hi ha molt contingut en xarxa que es pot aprofitar. No només teoria i tutorials experimentals, sinó models perfectament entrenats. Un model de visió per computador entrenat per detectar gats, pot ser molt útil per detectar gossos. Cal adaptar el model i entrenar-lo per a la seva finalitat, però pot estalviar molt de temps i esforç. Aquest concepte s'anomena transferència de coneixements.

La transferència de coneixements és possible perquè gran part dels nivells d'un model estan entrenats per detectar característiques generals de l'input que se li introdueix. Per exemple, els primers nivells d'un identificador de gats s'especialitzen en la detecció de línies verticals, horitzontals, obliqües... que tant serveixen per detectar gats com gossos. Son els últims nivells de la xarxa neuronal que s'especialitzen específicament en la detecció de gats.

És recomanable utilitzar la transferència de coneixements quan no es disposi de suficients dades per entrenar una xarxa neuronal de zero. En aquets casos s'haurà de buscar un model que comparteixi part de les característiques bàsiques de les dades. Per suposat, cal que tinguin el mateix tipus d'input. Ja sigui imatge, veu, text...

En funció del grau de connexió entre els dos projectes caldrà tornar a entrenar l'algoritme de forma més profunda o menys. Si es pot aprofitar la gran part de la xarxa neuronal excepte l'últim nivell, s'anomena *fine tunning.* Si el model original només serveix per aprofitar els valors inicials dels paràmetres, s'anomena *pre-training.*

3.12. Xarxes de convolució

3.12.1. Visió per computador

El camp de la visió per computador engloba diferents especialitats. La classificació i la classificació amb localització estan especialitzades en el tractament d'imatges amb un sol objecte. Per altra banda, la detecció d'objectes i la segmentació d'objectes estan especialitzades en el tractament d'imatges amb més d'un objecte.

La classificació d'imatges tracta d'identificar quina és la classe de l'objecte de la imatge. La classificació amb localització, afegeix un rectangle que delimita on es troba l'objecte. El

rectangle que delimita l'objecte rep el nom de *bounding box.*

La detecció d'objectes fa la classificació i localització d'imatges on hi apareixen múltiples objectes de múltiples classes. Per últim, la segmentació d'objectes també classifica i localitza imatges on hi apareixen múltiples objectes de múltiples classes però delimita el contorn de cada objecte en comptes de dibuixar-hi un rectangle.

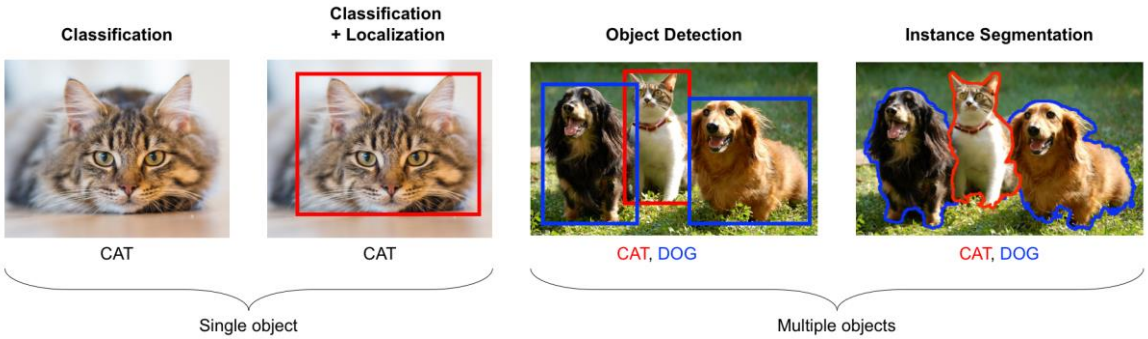


Fig 4 Tipus de visió per computador [7]

3.12.2. Necessitat d'una arquitectura específica pel tractament d'imatges

Les imatges contenen una gran quantitat d'informació, fet que obliga a una major complexitat de les xarxes neuronals convencionals dissenyades pel tractament d'imatges. Una imatge de 1000 x 1000 píxels x 3 canals de color, suposa la necessitat de tenir 3M de nodes al *input layer.* La dimensió d'una xarxa neuronal convencional que conté tants nodes al primer nivell és molt elevada i necessitaria moltes dades per ser entrenada. Per solucionar aquest problema, apareixen les xarxes neuronals de convolució.

3.12.3. Edge detection

Una de les característiques destacables de les xarxes de convolució és que analitza les imatges per grups de píxels propers. Es busquen relacions entre petites zones de les imatges en comptes de connectar la informació de tots els píxels alhora. En una xarxa neuronal convencional, cada node del segon nivell connectaria tots els nodes del primer nivell. A les xarxes de convolució, els nodes del segon nivell connecten només aquells píxels que estan pròxims entre si.

L'objectiu dels primers nivells és detectar els contorns dels objectes de la imatge, els "*edges".* Aquests *edges* són els límits entre, per exemple, el gat i el fons de la imatge. Com que el nombre de píxels observats pertany a una part molt petita de la imatge, els límits dels objectes s'observen com línies verticals horitzontals i obliqües.

Per exemple, suposem que es volen detectar límits verticals. Per fer-ho s'aplica un filtre a la

imatge. Un filtre és una matriu de dimensions reduïdes que escombra totes les posicions de la imatge i retorna un valor per cada posició que ocupa. El filtre 3x3 de la figura següent pot ocupar 5x5 posicions d'un input de 7x7.

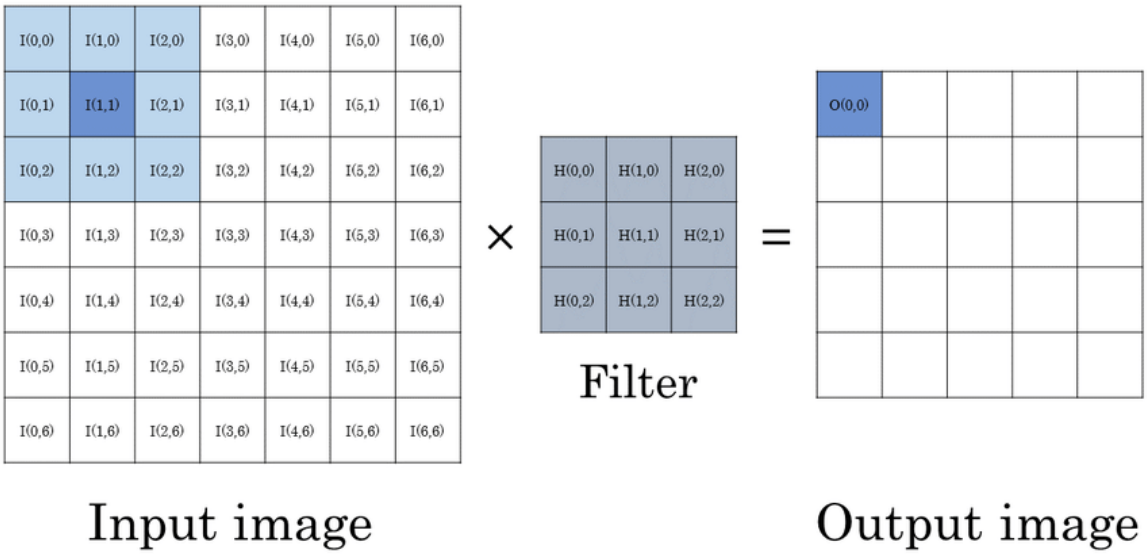


Fig 5 Aplicació d'un filtre 3x3 a una matriu 7x7 [8]

Un filtre vertical té la següent estructura:

1	0	-1
1	0	-1
1	0	-1

Fig 6

Aquesta estructura retornarà el valor 0 en zones de color uniforme i retornarà valors positius o negatius en zones que presentin diferències de color verticals. El signe dependrà de si el límit vertical està format per colors clars a l'esquerra i foscos a la dreta o al revés.

Els valors dels filtres es poden fixar de forma manual o es poden deixar com paràmetres que lliures a determinar a través de l'entrenament de l'algoritme.

3.12.4. *Padding*

Un dels problemes d'aplicar filtres per convolució és que el resultat de la convolució és una matriu més petita que l'anterior. L'altre inconvenient és que la informació dels límits de la

imatge s'utilitza menys que la informació del centre perquè el filtre utilitza els píxels centrals més vegades.

La solució d'aquests dos problemes és el *padding*. El *padding* consisteix en incrementar el nombre de píxels de la imatge, fer la matriu més gran. D'aquesta manera es solucionen els dos problemes comentats anteriorment. S'aconsegueix donar major pes als píxels que limiten la imatge alhora que es manté la mida de la imatge inicial.

3.12.5. *Strided Convolutions*

Són convolucions que no avancen de píxel en píxel, sinó que fan salts de "s" píxels. Això redueix més la mida de la matriu resultant ja que s'aplica menys vegades el filtre. Per calcular la mida de la matriu resultant en funció del *padding* i el *stride* utilitzem la següent fórmula:

$n \times n$ mida de la matriu

f x f mida del filtre

p *padding*

s *stride*

$$\left[\frac{n+2p-f}{s}+1\right] x \left[\frac{n+2p-f}{s}+1\right]$$

Eq 8

3.12.6. Convolució sobre volums

Les imatges a color tenen més d'un canal per cada píxel. Per exemple, les imatges codificades en format RGB tenen 3 canals, un corresponent a cada color. Per fer convolucions en imatges amb múltiples canals cal tenir tants filtres com canals tingui la imatge. Si es treballa en format RGB, cada convolució necessitarà 3 filtres. El resultat d'una convolució en volum és una matriu sense volum perquè els valors de les convolucions de cada canal es sumen. Suposant una matriu de 4x4 amb 3 canals en convolució amb filtres de 3x3 amb 3 canals, el resultat seria una matriu de 2x2 amb un sol canal.

Per incrementar la complexitat de la xarxa neuronal s'apliquen diversos tipus de filtre de la mateixa mida a cada nivell neuronal. Per exemple, al primer nivell es poden aplicar filtres de 3x3 que busquin *edges* verticals, horitzontals, oblics a 20 graus, oblics a -20 graus...

Cadascuna de les orientacions representa un filtre diferent. El nombre de filtres diferents d'un nivell és el nombre de canals de l'input del següent nivell.

3.12.7. *Bias* i funció d'activació

Als punts anteriors només s'ha explicat la part de la multiplicació de convolució de les xarxes neuronals de convolució. Aquestes xarxes, al igual que les convencionals, també incorporen un *bias* i una funció d'activació.

Un cop feta la multiplicació de la matriu d'entrada amb els filtres i obtingudes les matrius resultants de canal únic, és quan s'aplica el *bias* i la funció d'activació. El *bias* es suma a tots els valors de la matriu per igual, però hi ha un *bias* diferent per cada matriu. Un cop sumat el *bias* s'aplica la funció d'activació també a cada valor de la funció.

La figura següent resumeix el procés seguit en un nivell de convolució, conceptes comentats als últims punts. També dona referències per al càlcul de les dimensions de cada etapa:

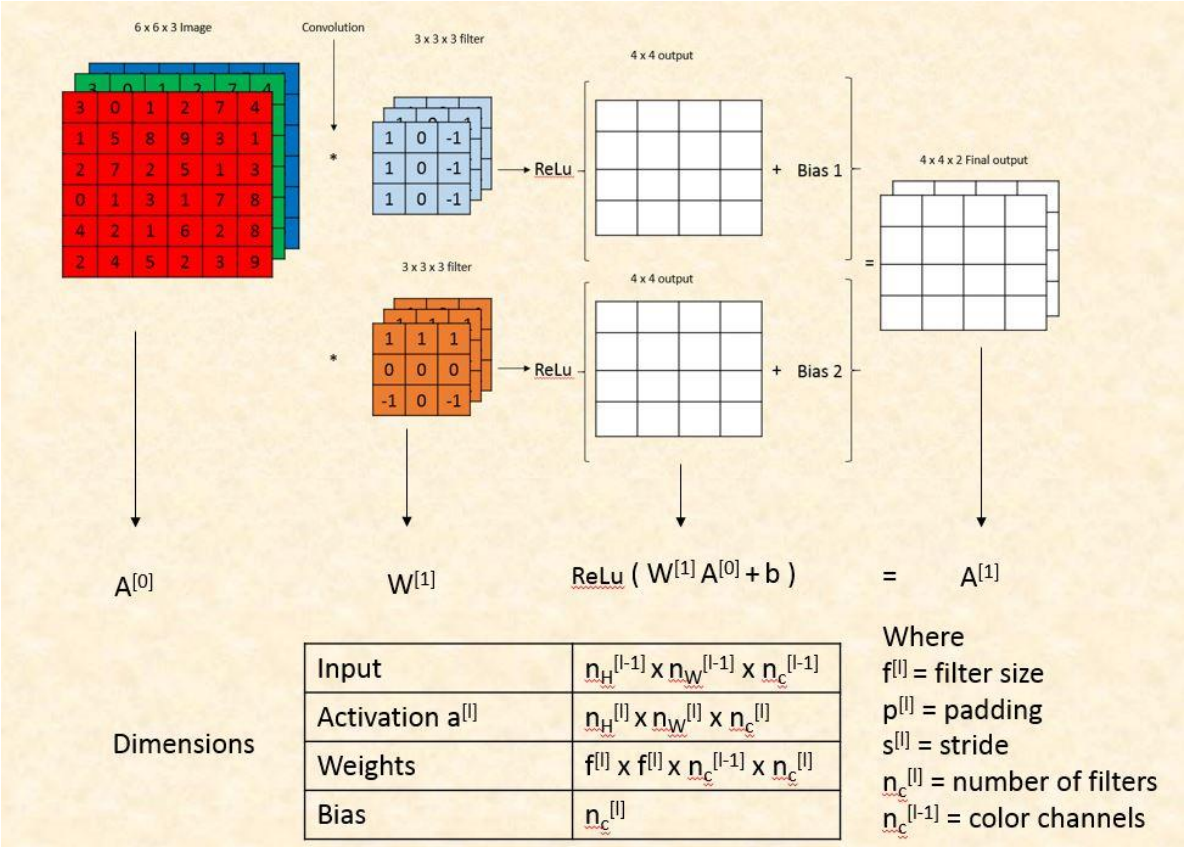


Fig 7 Dimensions d'un nivell de convolució [9]

3.12.8. *Pooling*

Dins d'una xarxa de convolució podem trobar altres nivells que no siguin de convolució. El *pooling* és un altre tipus de nivell. Consisteix en resumir la informació continguda a les diferents regions d'una matriu. Per fer-ho estudia diferents regions i obté el valor màxim, el mínim o la mitjana de tots els valors d'aquella regió.

El seu funcionament és igual al d'un filtre de convolució, recorre les diferents posicions d'una matriu i retorna un nombre per cadascuna d'aquestes posicions que ocupa. Per tant, cal triar la seva mida i el *stride* com a *hyperparameters.* La gran diferència amb els filtres de convolució és que no requereix paràmetres d'aprenentatge.

3.12.9. Exemple de xarxa de convolució

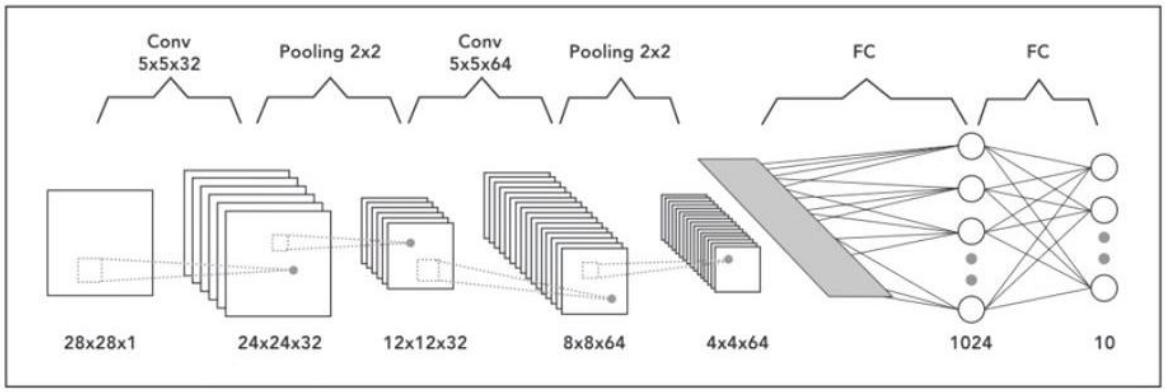


Fig 8 Arquitectura d'una xarxa de convolució [9]

L'arquitectura de la figura anterior podria ser l'exemple d'un model de reconeixement numèric. Donada una imatge amb un número del 0 al 9, és retorna quin d'aquests números apreix a la imatge. És per això que l'output te 10 nodes, cadascun d'ells retornarà la probabilitat de que sigui un dels números del 0 al 9. Per altra banda s'observa que l'input és de un canal com podria ser una imatge en blanc i negre.

D'entre les differents *hidden layers* trobem nivells de convolució, nivells de *pooling* i nivells totalment connectats. Observem com els primers nivells són de convolució i pooling per extreure característiques bàsiques de la imatge com podrien ser els *edges.* En canvi els últims nivells estan totalment connectats per agrupar la informació de tota la imatge i poder extreure informació a nivell global.

3.12.10. Avantatges de la convolució

Les avantatges de les xarxes de convolució són la reducció del nombre de paràmetres i la reducció de les dependències entre nodes de nivells diferents.

El que fa possible la reducció del nombre de paràmetres és l'ús dels filtres. Perquè un mateix filtre serveix per tota la imatge. Per una imatge de 1000x1000 píxels només calen 25 paràmetres si utilitzem un filtre de 5x5. Utilitzant una xarxa convencional necessitariem 1M de paràmetres.

L'ús dels filtres també ens permet reduir les connexions entre els outputs i els inputs del mateix nivell. Si tenim filtres de 3x3, cada valor de la matriu de sortida només depen de 9 valors de la matriu d'entrada. En canvi, en un nivell totalment connectat, cada valor de sortida depèn de tots els valors d'entrada.

Aquestes dos característiques simplifiquen les estructures neuronals i acceleren l'aprenentatge dels algoritmes.

3.12.11. Interpretació de les xarxes de convolució

Una bona manera de visualitzar què aprenen els diferents nivells de la xarxa de convolució és anar a buscar els píxels de les imatges que maximitzen una unitat d'un cert nivell. El que observem és que als nivell inicials es busquen característiques bàsiques de mida petita i que a mida que avancem, les característiques que maximitzen una unitat neuronal són més complexes i representen més píxels de la imatge. El fet de que representin més píxels de la imatge és per una causa estructural, ja que a mesura que avanço en els nivells de l'algoritme cada cop la mida de les matrius és més petita. Per tant, cada posició de les matrius resumeix la informació d'una part més gran de la imatge. El fet de que la informació continguda sigui més complexa a mesura que avanço en els nivells, és la conseqüència de que s'està agrupant totes aquestes característiques simples. [10]

3.13. Detecció d'objectes

3.13.1. *Sliding window detection algorithm*

Sliding window detection és un algoritme de detecció d'objectes que consisteix en un rectangle de mides constants que va "lliscant" per la imatge per analitzar si cadascuna d'aquestes regions conté un objecte d'interès o no. Sobre cada regió proposada s'aplica un classificador d'imatges que prediu quin objecte conté la regió. Aquest procés es repeteix per diferents mides i formes de rectangle per detectar objectes diversos.

L'inconvenient d'aquest mètode és l'alt cost computacional perquè ha d'estudiar molts requadres de la imatge que no tenen interès..

3.13.2. Implementació convolucional del *sliding window detection algorithm*

Com a solució del problema computacional va dissenyar un algoritme que aplicava la mateixa recerca per regions, però a través d'una convolució. D'aquesta manera es feien tots els càlculs al mateix temps i no de forma seqüencial. A més a més es compartia la complexitat computacional entre les diferents regions. Això va ser una millora per la comunitat de detecció d'objectes perquè feia el procés molt més eficient. [11]

Tot i això, aquest algoritme segueix tenint un inconvenient. Les *bounding boxes* no són gaire acurades ja que es generen de forma automàtica sense considerar on es troba l'objecte.

3.13.3. Intersecció sobre la unió

La detecció d'objectes comentada als punts anteriors es fa a partir d'una proposta automàtica de *bounding boxes* a partir de la qual es decideix si aquella regió conté un objecte o no. Per tant, és molt difícil que la *boundig box* predita per l'algoritme coincideixi a la perfecció amb la *bounding box* real. Així doncs, com es fa per donar per bones aquelles *bounding boxes* que delimiten correctament però no coincideixen amb la real?

Aquest és l'objectiu de la intersecció sobre la unió (IoU). És una operació que computa la intersecció entre el *bounding box* ideal i el *bounding box* de predicció.

$$IoU = \frac{intersecció\ de\ bounding\ boxes}{unió\ de\ bounding\ boxes}$$

Eq 9

Quan la IoU és major a un valor llindar, es dona per bona la predicció. Com més precisa hagi de ser la *bounding box,* més exigent haurà de ser el valor llindar.

3.13.4. *Non-max suppression*

El fet de provar tantes possibilitats de *bounding box* sobre la mateixa imatge fa possible la detecció múltiple pel mateix objecte. És possible que dos *bounding box* hagin detectat el mateix objecte amb una IoU major al llindar. Com podem fer-ho per que cada objecte només es detecti una vegada?

Non-max suppression és l'algoritme que tria el *bounding box* que millor s'ajusta a cada objecte. El procés que segueix per fer-ho és el següent. En primer lloc agafa el *bounding*

box amb la probabilitat més alta de contenir un objecte i busca totes les *bounding boxes* amb una IoU elevada amb la de màxima probabilitat. Totes aquelles *bounding boxes* amb una IoU major a un llindar s'eliminen perquè es considera que estan detectant el mateix objecte. Després pren la següent *bounding box* amb major probabilitat de contenir un objecte i repeteix el procés.

3.13.5. *Anchor boxes*

L'algoritme *non-max supression* elimina la possibilitat de detecció d'objectes en superposició. Com es poden detectar dos objectes diferents en superposició? Les *anchor boxes* són una possible solució perquè permeten detectar objectes diferents centrats en una mateixa regió de la imatge.

Per fer-ho es divideix la imatge en diverses cel·les i es proposen diferents *anchor boxes* per cadascuna d'aquestes cel·les. Cadascuna d'aquestes *anchor box* dona la possibilitat de detectar un objecte diferent. Les formes dels *anchor boxes* es poden triar de forma arbitraria o deixar com a paràmetres d'entrenament.

La mida dels *anchor boxes* és molt important ja que si no s'adequa als objectes de detecció és impossible que l'algoritme millori el seu rendiment. [12]

4. Software

Pel desenvolupament del projecte s'ha utilitzat un programari per facilitar les diverses tasques.

4.1. Etiquetat

4.1.1. LabelImg

LabelImg és una eina gràfica d'etiquetat d'imatges. LabelImg genera les *bounding boxes* de les imatges en format PASCAL VOC i guarda la informació a un fitxer XML. El programa està escrit en Python i és gratuït. [13]

4.2. Entorn de desenvolupament

4.2.1. Jupyter Notebook

Jupyter (Project Jupyter 2020) és un projecte de codi obert sense ànim de lucre que va néixer al 2014. Es fa servir com suport en el procés de programació de la ciència de les dades en tots els llenguatges de programació.

En concret s'ha utilitzat la Jupyter Notebook, que és una aplicació web que permet crear i compartir documents que contenen línies de codi, equacions, visualitzacions i text. A més a més del *machine learning* també s'utilitza per fer simulació numèrica, models estadístics, visualització de dades, transformació de dades, entre d'altres.

4.2.2. Google Colaboratory

Colaboratory és un producte de Google Research. Colab és una plataforma online basada en Jupyter que permet treballar en remot sobre els servidors de Google. L'eina ofereix la possibilitat de connectar-se gratuïtament a unitats de processat central (CPU), unitats de processat gràfic (GPU) o unitats de processat tensorial (TPU). Una de les avantatges de Google Colaboratory és que no cal instal·lar el programa ni llibreries de forma local. L'inconvenient és que requereix connexió a internet. [15]

4.3. Llenguatge de programació

4.3.1. Python

Python és un llenguatge de programació creat per Guido van Rossum l'any 1991. El llenguatge es desenvolupa sota una llicència de codi obert que permet la seva ràpida evolució gràcies a les aportacions de la comunitat de programadors que l'utilitza. El llenguatge conté milers de llibreries que faciliten la seva comprensió i milloren el seu rendiment.[16]

4.3.2. TensorFlow

Tensorflow és una llibreria especialitza en la creació de models de *machine learning.* Permet el desenvolupament i l'entrenament de models en llenguatge Python, JavaScrip o Swift. Tensorflow ofereix diversos nivells d'abstracció en funció de les necessitats dels usuaris. Per principiants en el món del *machine learning* existeix Keras, una API d'alt nivell.[17]

Una de les eines que brinda TensorFlow és TensorBoard, que permet fer un seguiment detallat dels resultats de l'algoritme durant el procés d'entrenament. Aquesta eina és molt útil per identificar com millorar l'algoritme.

4.3.3. Keras

Keras és una API per xarxes neuronals basada en Tensorflow i Python. La seva finalitat és adaptar la complexitat de Tensorflow a un nivell que permeti el seu ús de forma senzilla.[18]

5. Base de dades

Per fer l'entrenament d'un model neuronal es necessita una base de dades de la qual el model pugui aprendre. En el nostre cas la base de dades està formada per imatges de gambes acompanyades d'arxius XML on s'especifica la posició de totes les gambes que apareixen a la imatge. Les etiquetes serveixen perquè l'algoritme tingui un feedback sobre la precisió de les seves prediccions i així vagi aprenent. Al llarg de l'entrenament l'algoritme aprèn a detectar patrons de forma progressiva. Els primers nivells aprenen a detectar els límits dels objectes i poc a poc va augmentant la complexitat de les deteccions juntament amb el nivell de profunditat de l'algoritme. L'últim nivell és l'encarregat de detectar les gambes.

El que es dedueix del funcionament d'un entrenament és que només els últims nivells del model estan especialitzats en la detecció de l'objecte en qüestió. Els primers nivells comparteixen gran part de la seva informació i es poden aprofitar per detectar objectes diferents. A partir d'aquí surt la possibilitat de crear models a partir de models prèviament entrenats. D'aquests models pre-entrenats s'aprofiten els primers nivells i només es torna a entrenar els últims nivells, els especialitzats en la detecció de la classe.

Aquest projecte ha volgut aprofitar l'existència de models pre-entrenats amb grans bases de dades i fer l'entrenament a partir d'un model pre-entrenat. Això és especialment útil si no es disposa d'una base de dades especialment gran.

A més a més de les imatges destinades a l'entrenament, cal reservar un grup d'imatges per a fer el *testing*. El *testing* és el procés en que s'estudia el rendiment del model entrenat, se'l posa a prova. Per fer-ho se li ensenyen imatges verges, que el model no ha vist mai, i se li demana que detecti les classes de la imatge. En funció del percentatge d'encert del *testing* es decideix continuar amb l'entrenament o donar-lo per acabat. És important remarcar que aquest grup d'imatges ha de ser totalment nou per al model, sinó ens donaria informació molt enganyosa. Pot semblar obvi, però quan treballes amb centenars o milers d'imatges és complicat comprovar si una imatge a format part de l'entrenament o no, per això cal fer la separació entre imatges d'entrenament i imatges de test el més aviat possible.

5.1. Obtenció de les dades

La obtenció d'una biblioteca d'imatges de precisió ha estat un dels reptes del projecte. L'inici del projecte va coincidir en el temps amb el confinament causat pel COVID-19. Això va limitar les possibilitats de tenir imatges directes de les llotges del peix i va obligar a començar el projecte amb imatges d'internet. Per tant, els primers entrenaments es van fer

amb imatges de *Google Imatges*, que no son gens representatives de les imatges que després es volen detectar. Tot i això, aquests primers entrenaments van servir per familiaritzar-se amb les eines d'etiquetatge i amb l'algoritme.

A mesura que va anar passant el temps i les condicions van millorar, es va disposar d'imatges de més qualitat procedents de la llotja. La qualitat de les imatges rebudes era molt alta però la quantitat d'imatges molt reduïda. Per això es va decidir fer *'data augmentation'*. Aquest procés fa referencia a l'augment del la base de dades a partir de petites transformacions de les imatges existents. D'aquesta forma podem combatre el problema de manca de quantitat d'imatges alhora que augmentem la variabilitat de l'aparença de les gambes.

5.2. Data augmentation

Les imatges de les que es disposa són arxius JPG. Sobre cadascuna d'aquestes imatges s'ha etiquetat les gambes amb *bounding boxes* i s'ha emmagatzemat aquest informació en arxius PASCAL VOC en format XML. Per tant, cada imatge te un arxiu associat amb la informació de les seves *bounding boxes*. L'objeciu del *data augmentation* és la creació de noves imatges a partir de les imatges existents. Per evitar haver d'etiquetar les noves imatges, feina que implica un elevat cost en temps, es vol generar els nous *bounding boxes* de forma automàtica a partir dels *bounding boxes* originals.

Per aconseguir-ho s'ha seguit una publicació publicada a la plataforma Medium: *Learn to Augment Images and Multiple Bounding Boxes for Deep Learning in 4 Steps* [19]

Les transformacions utilitzades són: re-escalat , rotació, translació, ampliació, *GaussianBlur* i *AdditiveGaussianNoise.* La unió de totes aquestes imatges ha acabat creant una base de dades de 2500 imatges.

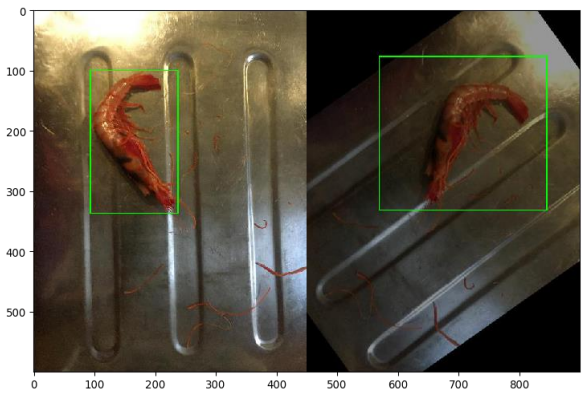


Fig 10 Rotació

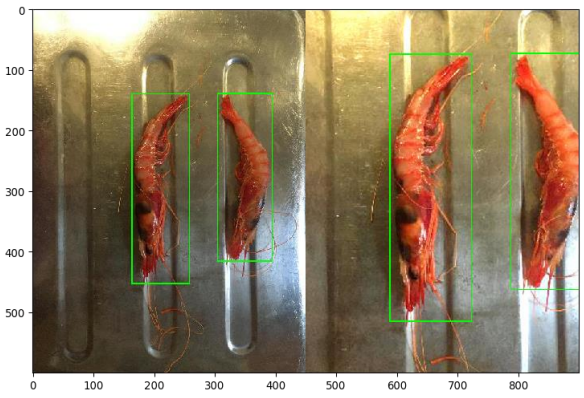


Fig 9 Ampliació

5.3. Unificat de mides d'imatge

Tot i que per entrenar una xarxa neuronal a través de TensorFlow Object Detection API la base de dades pot contenir imatges de diferents mides, és recomanable controlar la mida de les imatges per evitar la saturació del GPU o CPU que ha de processar les dades. Per això totes les imatges s'han re-escalat a una mida de 600 x 600.

6. Algoritmes

6.1. Faster R-CNN

La Faster R-CNN neix de la modificació de xarxes preexistents. En aquest cas, a partir de la *Region-Based Convolutional Neural Network* o R-CNN (Liang y Hu 2015). La R-CNN és una xarxa dedicada a la detecció d'objectes. Com hem vist, la detecció d'objectes te dos objectius. El primer és la localització de l'objecte dins de la imatge i el segon és la classificació. Comparat amb la classificació d'objectes, la detecció d'objectes requereix mètodes més complexes de resolució.

Per arribar a la detecció d'objectes, la R-CNN extreu al voltant de 2000 regions d'interès (RoI) i les analitza per separat per classificar-les. Les RoI són regions proposta on l'algoritme preveu que podria haver-hi un objecte. L'inconvenient d'aquesta xarxa és que degut a la seva complexitat és lenta e inelegant.

La primera millora del R-CNN va ser el Fast R-CNN. Aquest mètode permet fer la classificació de totes les regions d'interès (RoI) al mateix temps i així accelerar la fase d'entrenament i test. Per entrenar una arquitectura molt densa com la VGG16, el mètode Fast R-CNN és 9x més ràpid que el R-CNN, 213x més ràpid en el test i assoleix una major precisió sobre la base de dades PASCAL VOC 2012. [21]

L'inconvenient de la Fast R-CNN és que requereix algoritmes previs de proposició de regions d'interès. Avenços com el Fast R-CNN redueixen el temps de classificació de les xarxes neuronals i posen en evidència les limitacions computacionals dels algoritmes de proposició de regions, reconeixent-los com a coll d'ampolla de la detecció d'objectes.

El Faster R-CNN ofereix una solució a aquest problema amb la introducció de les *Region Proposal Networks* (RPN), encarregades de la proposició de regions. Les RPN comparteixen els nivells de convolució amb la xarxa de detecció d'objectes. El fet de compartir les convolucions fa que es redueixi a quasi zero el cost computacional de la proposició de regions en comparació amb el cost de la classificació. El resultat és una xarxa que és capaç de detectar 5 imatges per segon en una GPU amb una precisió del 70.4% mAP sobre la base de dades PASCAL VOC 2012.

El funcionament de la RPN es basa en una aplicació en convolució del mètode del *sliding window.* Per cada posició del *sliding window* s'estudien 9 *anchor boxes,* 3 mides de *anchor box* (128^2, 256^2 i 512^2) amb 3 proporcions diferents (1:1, 1:2 i 2:1). Com a resultat de la RPN s'obté la probabilitat de la presència d'un objecte i la probabilitat de la no presència

d'un objecte per totes les *anchor boxes* de la imatge. Per entrenar una RPN s'assigna un valor binari a cada *anchor box,* es computa una funció *loss* i s'optimitzen els paràmetres per *backpropagation* amb mètodes com el *gradient descent*.

Per poder unir una RPN amb una Fast R-CNN de forma que comparteixin nivells de convolució es segueixen 4 passos. En primer lloc s'entrena la RPN com s'ha descrit anteriorment. S'inicialitza amb un model ImageNet i s'optimitzen els paràmetres. En el segon pas s'entrena per separat una xarxa Fast R-CNN utilitzant les propostes generades al primer pas. Els paràmetres d'aquesta xarxa també s'inicialitzen amb un model ImageNet. En aquest punt les dues xarxes encara no comparteixen nivells de convolució. En el tercer pas, s'utilitza la xarxa Fast R-CNN per inicialitzar l'entrenament de la RPN, però es fixen els nivells de convolució compartits i només s'optimitzen els paràmetres que pertanyen únicament a la RPN. En aquest pas ja comparteixen nivells de convolució. Finalment, mantenint els nivells de convolució compartits, s'optimitzen els nivells de convolució de la Fast R-CNN. [22]

6.2. Inception V2

Al igual que la Faster R-CNN, la Inception V2 també és l'evolució d'una xarxa anterior, la Inception. Inception va sorgir per fer front al ImageNet Large-Scale Visual Recongnition Challenge 2014. El fet més destacable d'aquesta xarxa és la seva arquitectura, que millora el rendiment dels recursos computacionals que utilitza. Aquesta arquitectura aconsegueix incrementar la profunditat i la amplada de la xarxa sense augmentar els recursos computacionals. Un exemple d'aquest tipus d'arquitectura és la GoogLeNet, una xarxa de 22 nivells que es va presentar per al ImageNet Large-Scale Visual Recognition Challenge 2014. GoogLeNet va deixar constància dels beneficis de l'arquitectura Inception amb el seu bon rendiment en comparació amb la resta de competidors.

Les xarxes de convolució estaven típicament estructurades amb nivells de convolució enllaçats uns amb altres i amb la possibilitat d'incorporar nivells *pooling* entremig. Investigadors d'aquest camp han desenvolupat variacions respecte aquesta estructura per millorar el seu rendiment. Alguns exemples són la LeNet, la Network-in-Network o la R-CNN ja comentada.

La manera més directe de millorar el rendiment d'una xarxa és incrementant la seva mida. Tant la seva profunditat amb el nombre de nivells com la seva amplada amb el nombre d'unitats a cada nivell. Aquesta orientació te dos inconvenients. En primer lloc, xarxes més grans suposen un nombre més gran de paràmetres, fet que augmenta la possibilitat d'*overfitting* si el nombre d'exemples d'entrenament és limitat. L'altre inconvenient és l'increment dels recursos computacionals necessaris. En una xarxa de dos nivells de

convolució, qualsevol increment del nombre de filtres suposa un increment quadràtic de computació. Cal assegurar-se de que l'increment de computació està tenint un impacte en els resultats per no malgastar recursos.

La manera de contrarestar aquests inconvenients és intercanviar els nivells totalment connectats per nivells menys connectats. L'arquitectura Inception segueix aquesta línia, busca la manera d'utilitzar components totalment connectades per aconseguir estructures vagament connectades. La xarxa està construïda a partir de blocs de convolució, però estan col·locats de tal forma que es redueix el cost computacional. Per fer-ho es combinen filtres 1 x 1, 3 x 3 i 5 x 5 amb operacions *pooling* en un mateix nivell. Les convolucions 1 x 1 s'utilitzen per reduir la complexitat dels inputs abans de les convolucions 3 x 3 o 5 x 5. Aquest tipus d'estructura permet incrementar el nombre d'unitats d'un nivell sense tenir un impacte incontrolat sobre la complexitat computacional. [23]

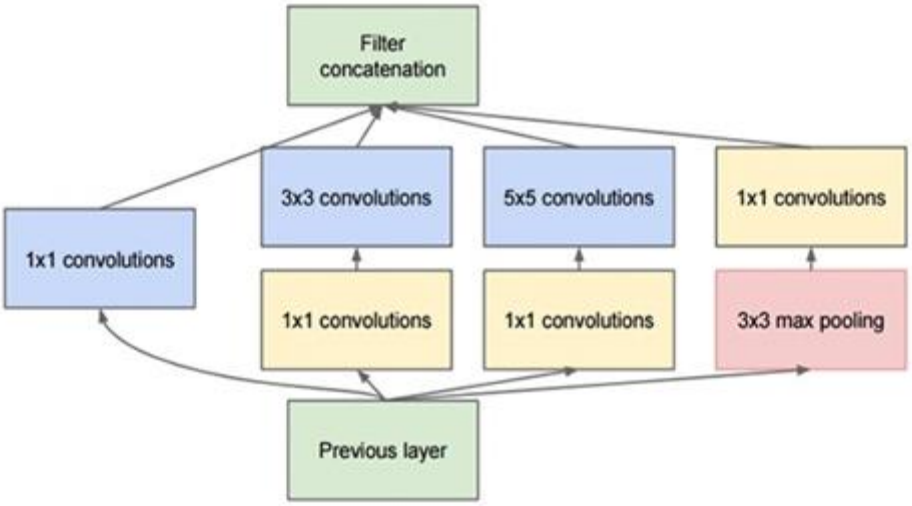


Fig 12 Mòdul unitari d'una arquitectura Inception [23]

Tot i que l'arquitectura Inception va millorar el rendiment d'altres arquitectures que perseguien el mateix objectiu, la seva complexitat feia difícil les adaptacions i canvis per a noves aplicacions. Inception V2 sorgeix amb l'objectiu de solucionar aquest problema de flexibilitat alhora que millora el rendiment de l'estructura.

La primera millora és la factorització dels filtres 5 x 5 en dos operacions 3 x 3 per augmentar la velocitat computacional. Una operació 5 x 5 és 2.78 vegades més costosa que una 3 x 3.

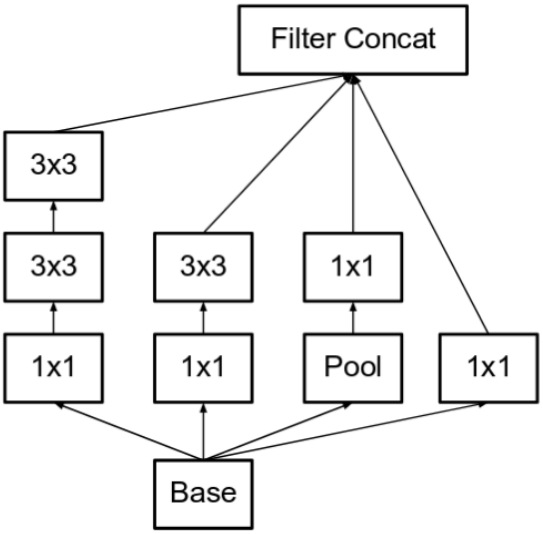


Fig 13 Mòduls Inception on es substitueixen les convolucions 5 x 5 per dos convolucions 3 x3 [24]

La segona millora és la factorització de les convolucions *n x n* a una combinació de 1 x *n* i *n* x 1. Una convolució de 3x3 és equivalent a una convolució 1 x 3 seguida d'una convolució 3 x 1. Aquest segon mètode estalvia un 33% del cost computacional.

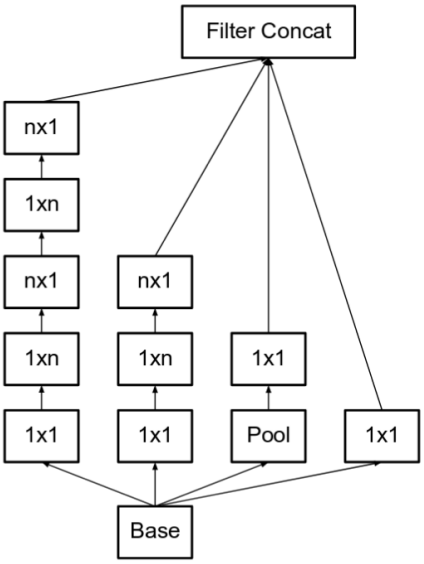


Fig 14 Mòdul Inception després de factoritzar les convolucions *n x n* [24]

La última millora és expandir la factorització en amplada en comptes de profunditat per evitar una excessiva reducció en les dimensions i una conseqüent pèrdua d'informació. La figura següent il·lustra aquesta expansió en amplada.[24]

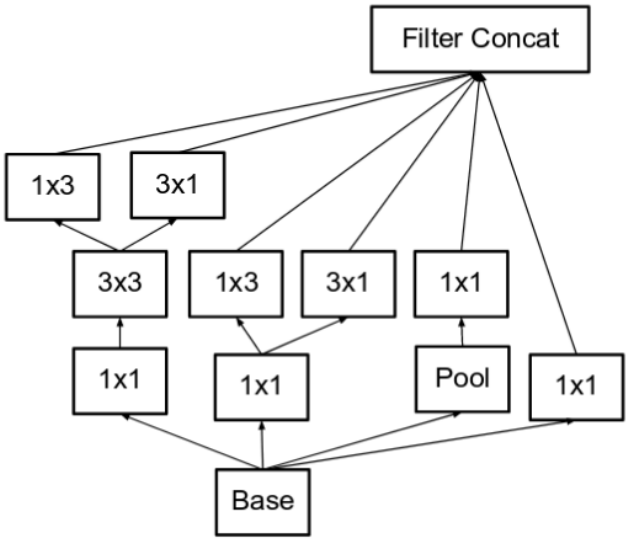


Fig 15 Mòduls Inception amb expansió de filtres en amplada[24]

7. Experimentació

7.1. Faster-R-CNN-Inception-V2

Per la implementació del Faster R-CNN s'ha utilitzat un model ja entrenat amb la base de dades COCO. Ens pot ser útil aprofitar l'entrenament dels nivells més profunds d'un altre model per la nostra aplicació.

Els resultats del primer entrenament han mostrat una falta de dades. Al visualitzar com evolucionava el *loss* del *training set* i del *dev set* a mesura que avançava l'entrenament, s'observa una diferència molt gran entre l'evolució dels dos. El ritme d'aprenentatge de l'algoritme és molt més alt sobre el *training set* que sobre el *dev set.* Per tant, el que s'observa és un problema de variància. Una solució per reduir la variància és augmentar la base de dades del *training set.* D'aquesta manera s'evita que el model s'especialitzi massa sobre les dades que entrena.

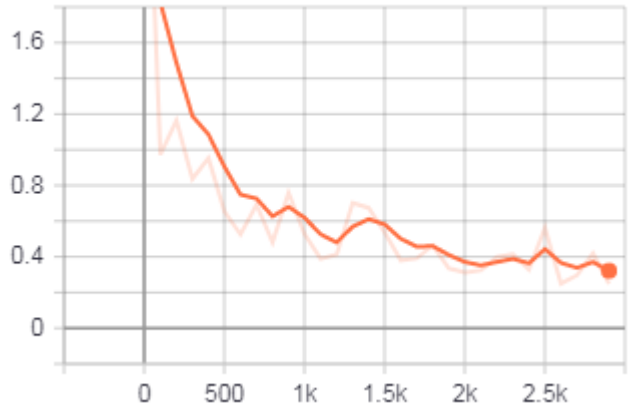


Fig 16 Loss del training set en funció dels steps d'entrenament

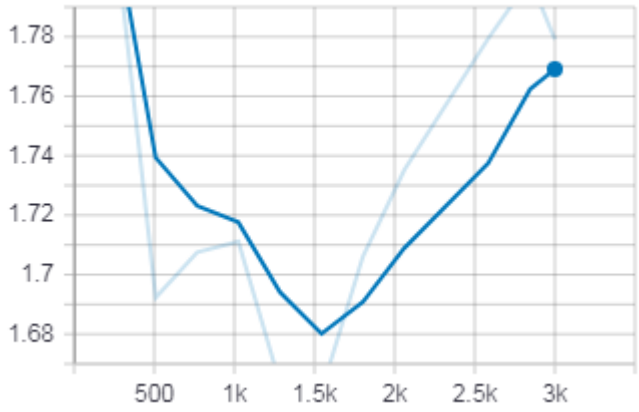


Fig 17 Loss del dev set en funció dels steps d'entrenament

Tal i com es pot observar als gràfics anteriors, la reducció del *loss* del *dev set* és pràcticament nul·la en comparació amb la del *training set.* Fins i tot s'observa com a partir del *step* 1.500 el *loss* comença a aumentar. La diferència de rendiment entre el *train* i *dev set* és la que quantifica la variància del model. El fet de tenir variància vol dir que estem entrenant el model específicament per un grup d'exemples i no s'està aconseguint extreure les característiques generals de l'aprenentatge. Aquest fenòmen també s'anomena *overfitting*.

Està clar que hi ha un problema de variància, però abans d'extreure conclusions cal analitzar bé perquè s'està donant aquesta gran diferència entre el *test* i el *dev set*. Per fer-ho estudiem el rendiment del model sobre les imatges del *dev set* per veure on està fallant i quin tipus d'imatge està creant aquest baix rendiment.

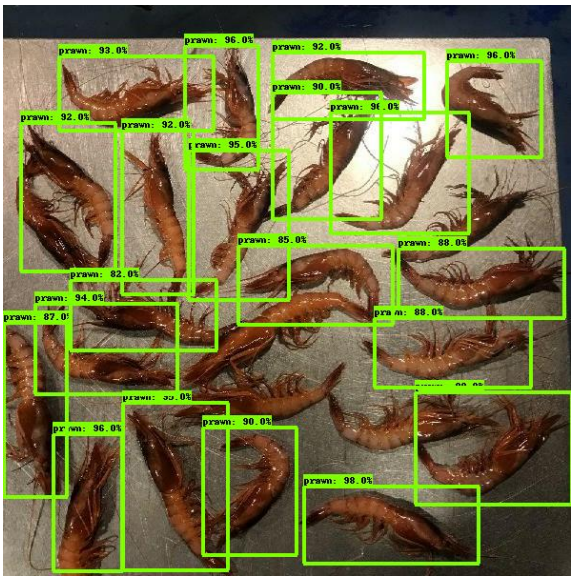


Fig 19 Dev_set 1

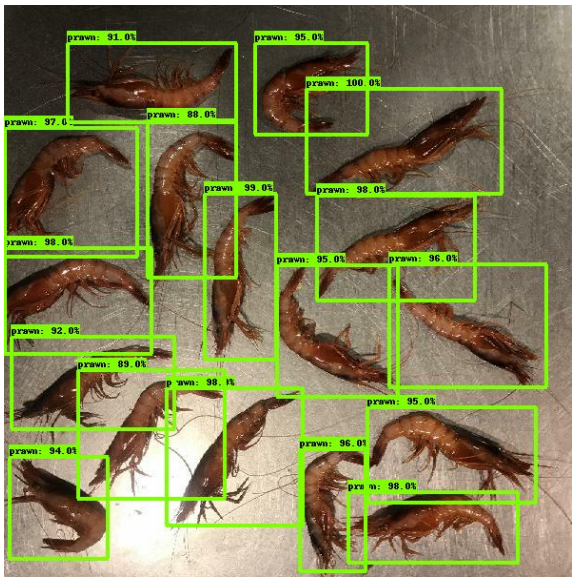


Fig 18 Dev_set 2

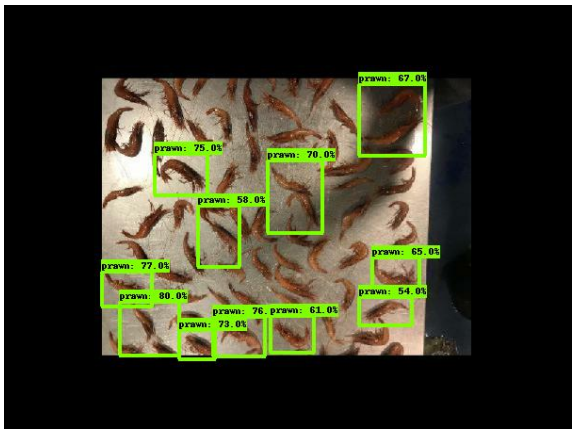


Fig 20 Dev_set 3

Imatge	Falsos Negatius	Falsos Positius	Precisión (%)	Recall (%)	#gambes a la imatge
Dev_set 1	4	0	100	87,5	25
Dev_set 2	0	0	100	100	16
Dev_set 3	47	0	100	18,9	58

Fig 21 Anàlisi resultats dev set

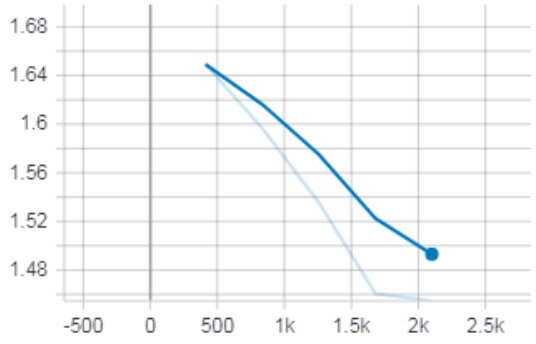
El primer que notem al estudiar el rendiment del *dev set* és que una sola imatge aglutina el 92% dels falsos negatius. Es pot intuir que això es deu al major nombre de gambes presents a la imatge i a la menor mida de cada gamba. Per tant, quan una gamba ocupa pocs píxels de la imatge costa més detectarla. Aquest fet es veu agreujat perquè la imatge Dev_set 3 és el resultat d'un *zoom out* degut al *data aumentation.*

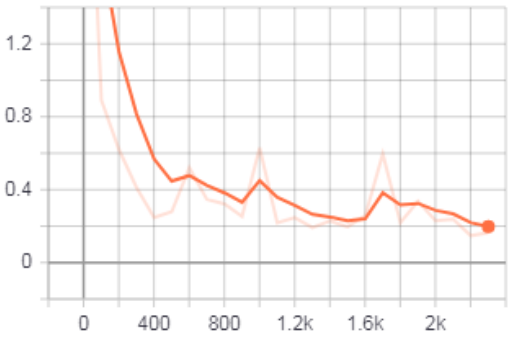
Una de les recomanacions dins del camp del *machine learning* és tenir un dev set amb la mateixa distribució que les imatges que es trobarà el model a l'aplicació final. És a dir, cal que les imatges del *dev set* s'assemblin el màxim possibles a les imatges que rebrà el model a l'aplicació final. Això no sempre és possible, però en el nostre cas sí que ho és. Per tant, és un error col·locar una imatge distorsionada pel *data aumentation* al *dev set* perquè estem evaluant el model sobre unes imatges irreals. En aquest cas específic, estem intentant que el model detecti gambes més petites del que es trobarà a posteriori a la realitat i per tant estem malgastant esforços en un objectiu que no aporta valor al projecte.

Per solucionar aquest problema cal redefinir el *dev set* i fer-lo el més realista possible perquè els resultats de l'entrenament ens donin informació fiable sobre com es comportarà el model a la realitat.

7.1.1. Millora del *Dev set*

Per tenir un *dev set* fiable s'han inclòs només imatges originals provinents de la llotja, les més properes a l'aplicació final. Un cop fet aquest canvi al *dev set* tornem a entrenar el model.





Els resultats mostren molt poca millora respecte al model anterior. En aquest cas no veiem un increment del *loss* de *dev set* a mesura que avança l'entrenament, però la reducció del *loss* del *dev set* representa menys d'un 10% del que es redueix el *loss* al *train set*.

Tornant a fer un estudi sobre les imatges del *dev set* per identificar en què falla el model, seguim veient el mateix comportament.

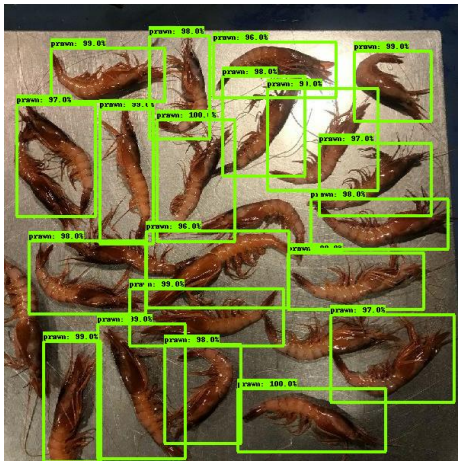


Fig 22 Dev_set 1

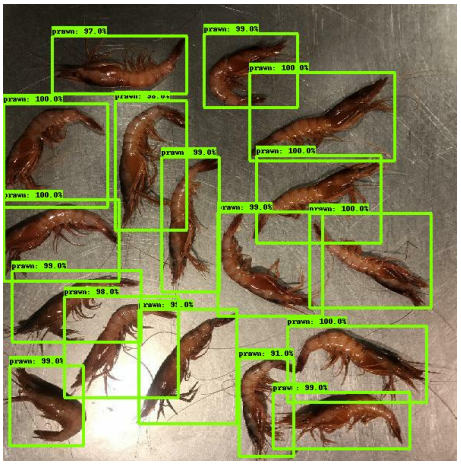


Fig 23 Dev_set 2

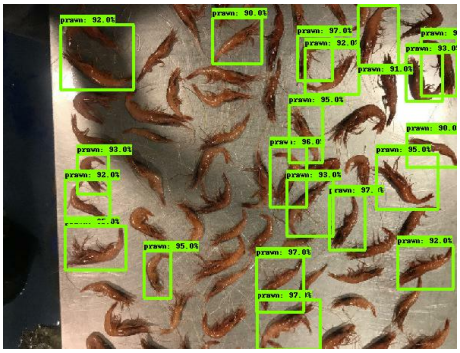


Fig 24 Dev set 3

Tenim una precisió del 100%, però el *recall* cau fins al 36,2% quan les gambes ocupen pocs píxels.

Imatge	Falsos Negatius	Falsos Positius	Precisión (%)	Recall (%)	#gambes a la imatge
Dev_set 1	4	0	100	87,5	25
Dev_set 2	0	0	100	100	16
Dev_set 3	37	0	100	36,2	58

Fig 25 Anàlisi resultats del Dev set

El fet de triar un *dev set* realista ha fet que l'exigència del model sigui més baixa i ha significat un millor rendiment. Cal remarcar que estem preparant el model per rendir amb un tipus d'imatge en concret. Intentar fer que rendís amb tot tipus d'imatge no aportaria valor i seria un contrasentit.

Alhora que hem vist la importància d'utilitzar un *dev set* similar a l'aplicació final, també hem identificat que al model li costa detectar les gambes de mida petita. Per tant l'objectiu és treballar per millorar en aquesta direcció.

7.1.2. Millora *anchor boxes*

La següent proposta que s'ha fet és canviar la mida dels *anchor boxes*. Les *anchor boxes* són les regions proposades per l'algoritme per detectar els objectes. Es tracta de rectangles on es pot variar tant la seva mida com la proporció entre base i alçada de forma arbitrària. En aquest cas s'ha disminuït la mida dels *anchor boxes* a la meitat i s'han mantingut les

proporcions de sèrie entre base si alçada.

L'objectiu d'aquest canvi era que la intersecció entre les *groundtruth boxes* i les *bounding boxes* proposades fos el més gran possible i per tant es detectessin més gambes petites.

El resultat de l'entrenament amb el canvi de mida de les *anchor boxes* no ha representat una millora significativa.

Tot i no observar-se una millora, sí que s'ha observat un tret molt peculiar. El model només és capaç de detectar 20 gambes per imatge com a màxim. Per aprofundir sobre aquesta observació s'ha fet altres entrenaments per estudiar si l'algoritme utilitzat té un límit de 20 deteccions per imatge. Aquest estudi ha resultat positiu, l'algoritme que s'estava utilitzant tenia una limitació de 20 deteccions per imatge, fet que limitava molt el rendiment dels models.

7.1.3. Millora límit de deteccions per imatge

Un cop canviada la configuració de l'algoritme que limitava el nombre de deteccions per imatge els resultats canvien molt. Les prediccions sobre les imatges del *test* detecten correctament el 95% de les gambes.

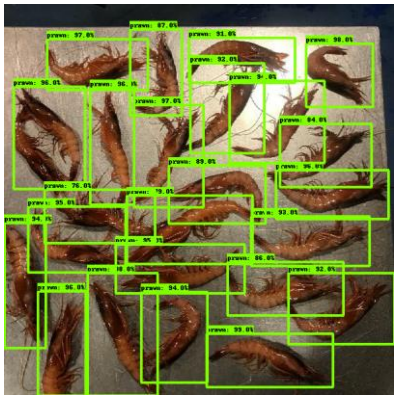


Fig 26 Dev_set 1

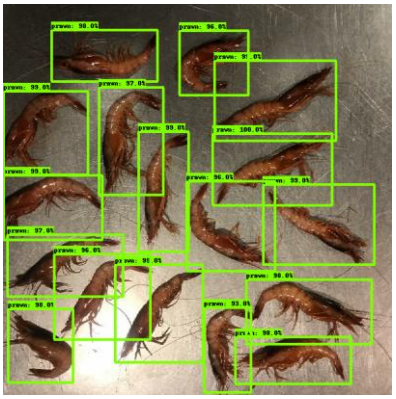


Fig 27 Dev_set 2

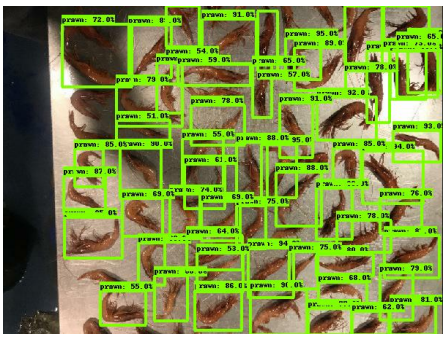


Fig 28 Dev_set 3

7.2. Faster R-CNN Resnet 50

Després de fer diversos experiments fallits amb el Faster R-CNN Inception-V2 , es decideix provar un altra variació del Faster R-CNN, la Resnet 50.

Els resultats d'aquest experiment mostren una forta variància entre les dades d'entrenament i les dades del *dev set*. Aquesta diferència es pot veure als gràfics que mostren el *loss* del *trainning set* i del *dev set* respectivament:

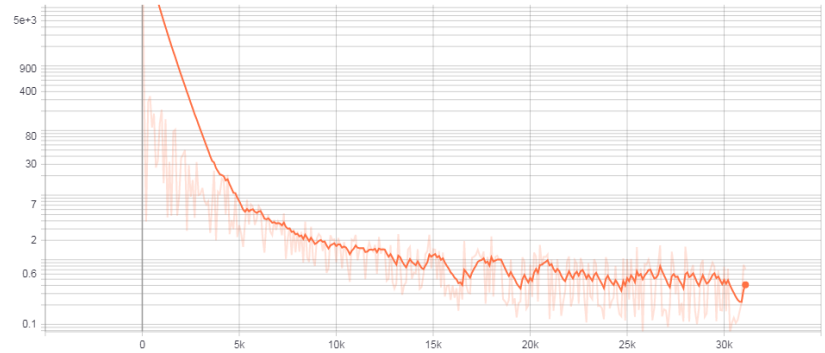


Fig 29 Loss del *trainning set*

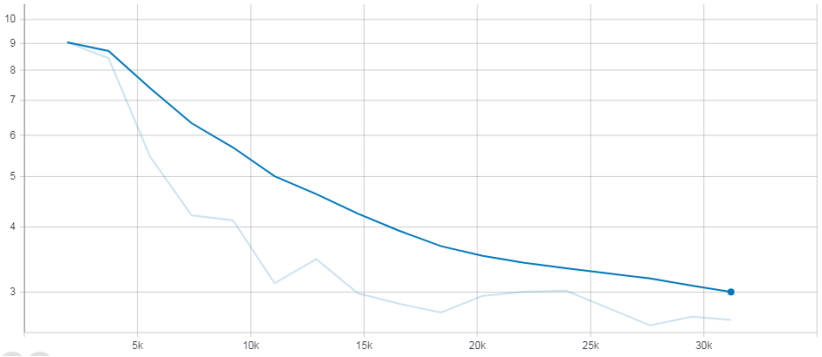


Fig 30 Loss del *dev set*

S'observa que el *trainning set* aconsegueix unes pèrdues de 0,4 al final de l'entrenament i el *dev set* es queda amb unes pèrdues mínimes de 2,6. Els resultats sobre les imatges són els següents:

Fig 31 Dev_set 1

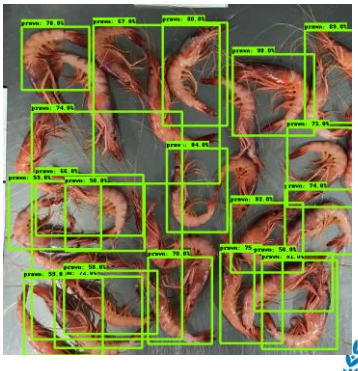


Fig 32 Dev_set 2

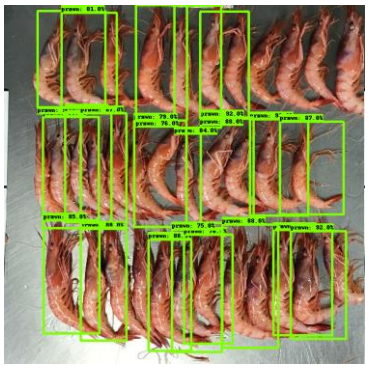


Fig 33 Dev_set 3

S'obté una alta precisió però un *recall* molt reduit, hi ha molts falsos negatius.

7.2.1. Reducció de la variància

Amb la intenció de reduir la variància s'augmenta la base de dades d'entrenament. D'aquesta manera costarà més que el model s'especialitzi sobre les dades d'entrenament.

S'ha afegit a la base de dades d'entrenament 440 imatges noves que son el resultat d'augmentar 40 imatges originals.

Els resultats d'aquest entrenament no han mostrat gaires diferències respecte el model anterior. Es maté la variància tot i l'augment considerable de les imatges.

A més a més d'existir una alta variància, els resultats sobre les imatges de validació no són suficientment bons. Hi ha un alt percentatge de falsos negatius i falsos positius.

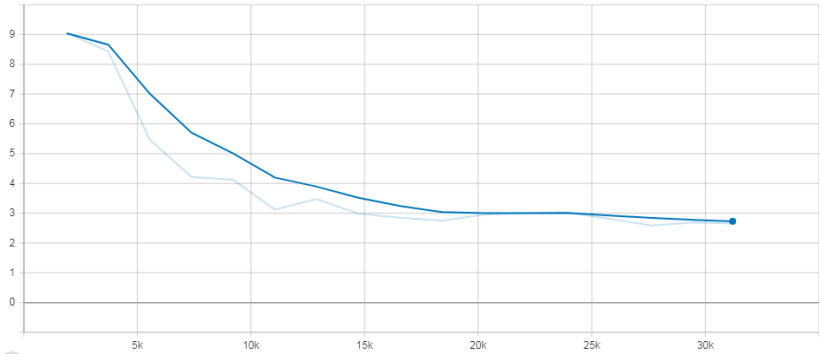


Fig 34 Validation loss

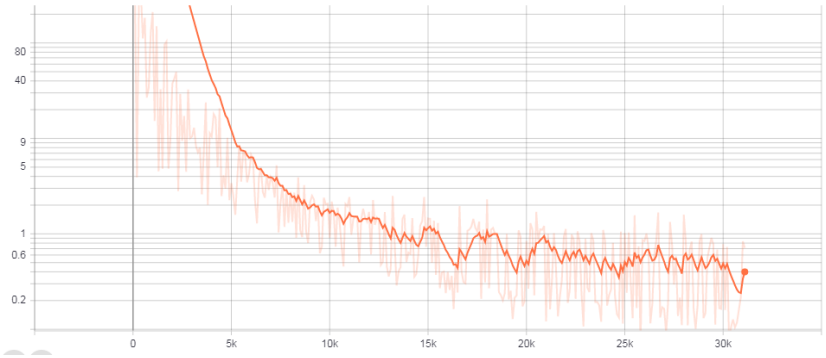


Fig 35Trainning loss

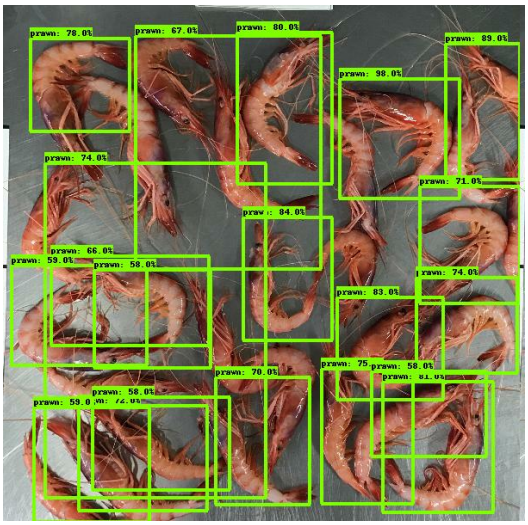


Fig 36 Imatge de validació

7.2.2. Nou increment d'imatges per reduir la variància

Amb la mateixa intenció que a l'experiment anterior, s'augmenta la base de dades d'entrenament amb 1900 imatges generades a través d'augmentar 190 imatges originals.

Tot i l'augment considerable de les imatges, es segueix sense veure un canvi a la variància del model.

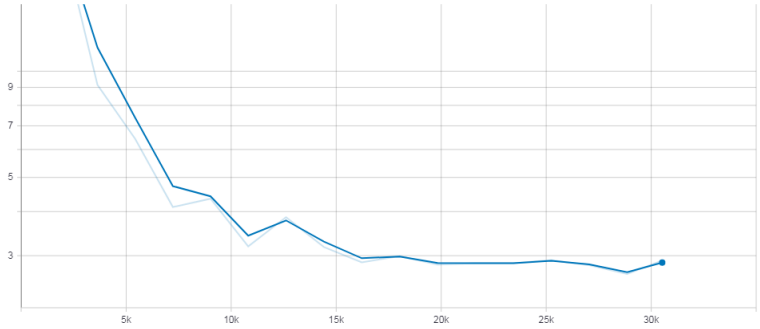


Fig 37 Validation loss

Fig 38 Trainning loss

7.2.3. Allargament de l'entrenament

En vista de que un increment en les imatges no tenia un impacte significatiu sobre els resultats, es decideix canviar d'estratègia. Per a la següent iteració es decideix augmentar el temps d'entrenament. Durant tot els entrenaments fets amb la Faster R-CNN Resnet 50 s'havia entrenat durant 25 *epochs.* A aquesta nova iteració s'allarga l'entrenament fins les 50 *epochs.*

Fent una valoració quantitativa, veiem com el model disminueix les pèrdues del *test set* i augmenten la precisió i el *recall*. Fent una valoració qualitativa veiem el model es comporta de forma molt precisa quan les gambes es mostren separades entre si i va perdent precisió a mesura que hi ha més gambes a la imatge.

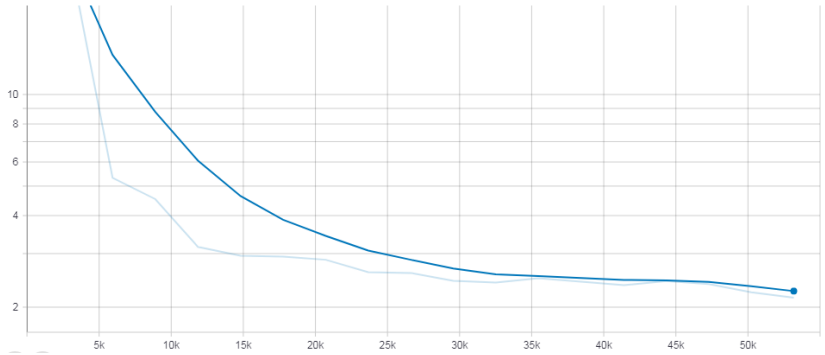


Fig 39 Validation loss

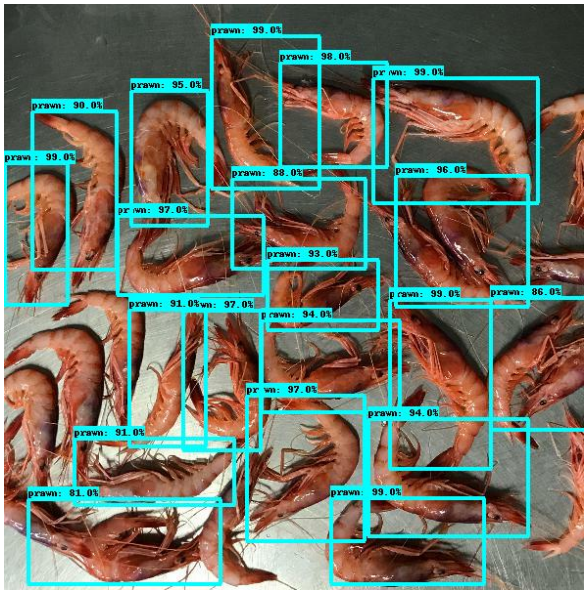


Fig 40 Imatge de validació

Als resultats es pot observar una alta precisió sobre les deteccions, però un *recall* baix. És a dir, les gambes que es detecten es detecten de forma precisa, però n'hi ha moltes que no es detecten. Reflexionant sobre com es podria millorar aquest aspecte s'ha considerat la possibilitat que el nombre de deteccions estigués limitat per l'algoritme. Al fer el recompte del número de deteccions màximes de totes les imatges s'ha vist que és 20. Per corregir aquesta limitació s'ha canviat la configuració de l'algoritme i s'ha fixat el màxim de deteccions a 100.

7.2.4. Canvi limitació de deteccions i allargament de l'entrenament

Un cop corregida la limitació es torna a entrenar l'algoritme fins a 75 *epochs* i els resultats han estat molt diferents. Aquest cop sí, s'aconsegueix una precisió i un *recall* elevats.

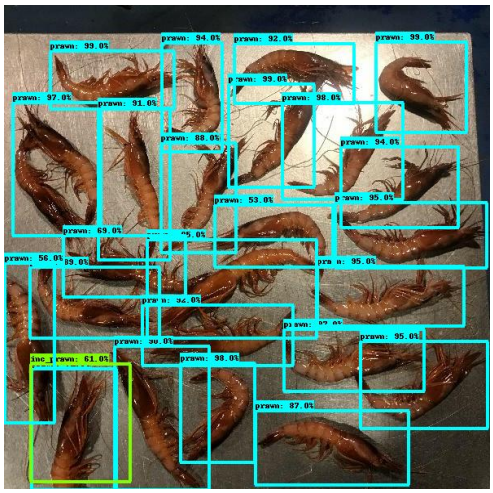


Fig 41 Imatge de validació 1

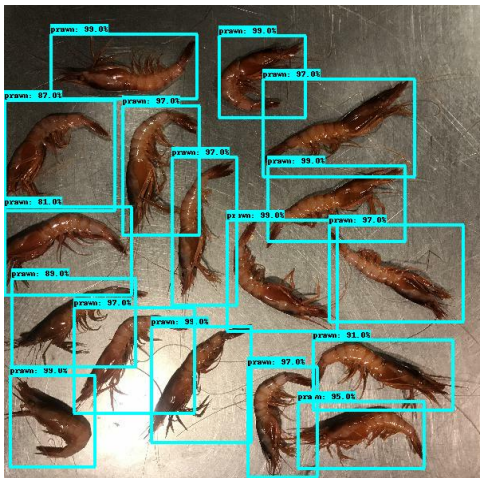


Fig 42 Imatge de validació 2

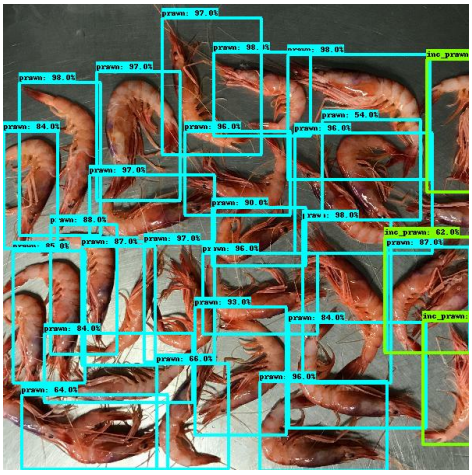


Fig 43 Imatge de validació 3

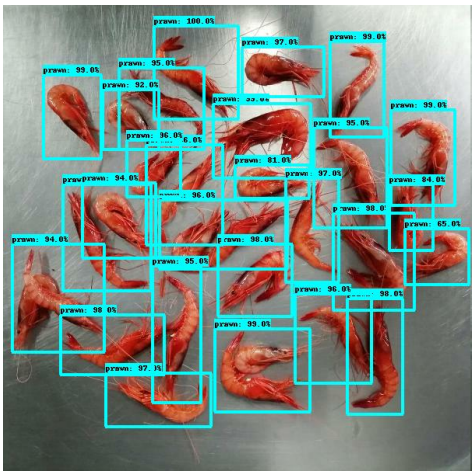


Fig 44 Imatge de validació 4

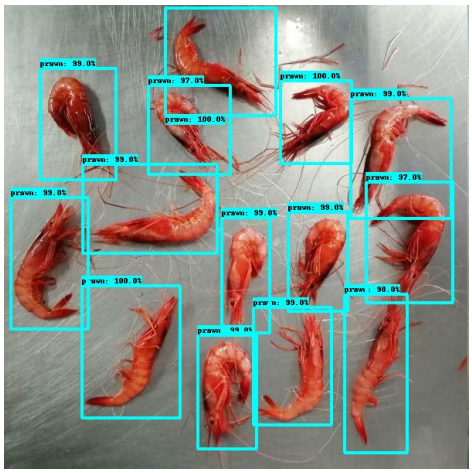


Fig 45 Imatge de validació 5

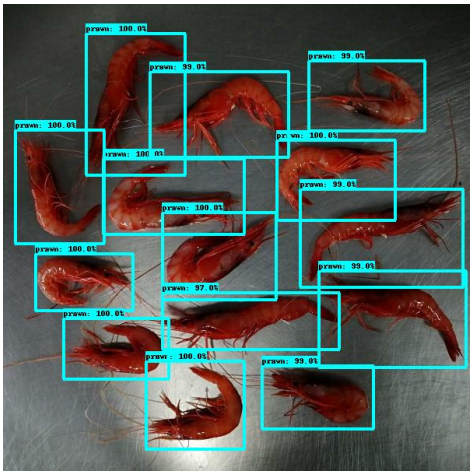


Fig 46 Imatge de validació 6

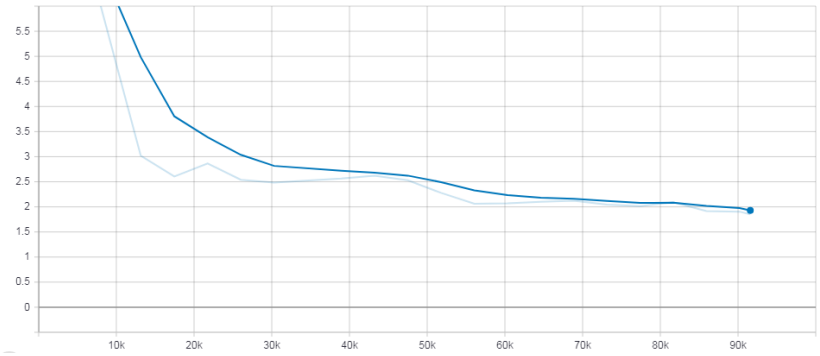


Fig 47 Dev loss

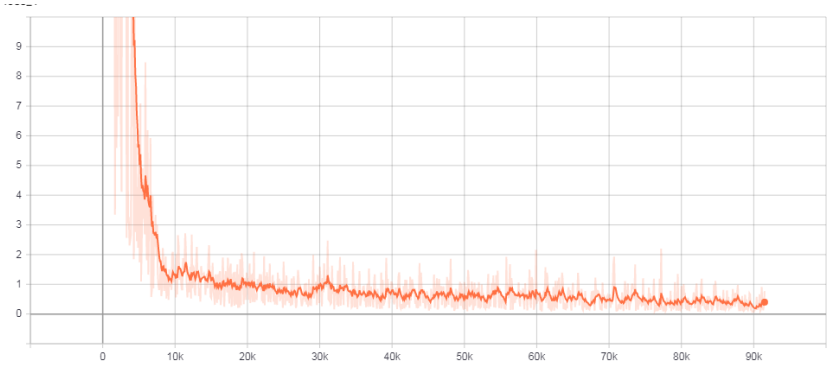


Fig 48 Train loss

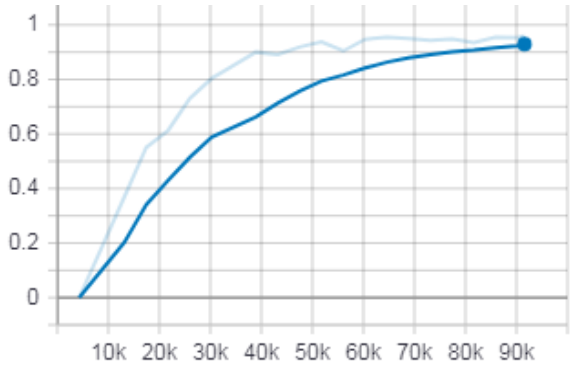


Fig 49 Average Precisión

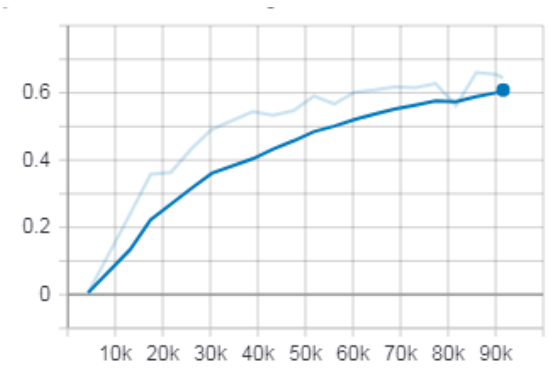


Fig 50 Average Recall

Pressupost del projecte

Cost del projecte				
Cost del Software i Aprenentatge	**Cost anual [€]**	**Cost de 5 mesos [€]**	**Us**	**Cost [€]**
Google Colaboratory PRO	120	50	100%	50
Microsoft Office Professional	600	250	80%	200
Hores d'aprenentatge (50 hores)				1000
TOTAL				**1250**
Cost de disseny i desenvolupament	**Hores [h]**	**Cost per hora [€/h]**		**Cost [€]**
Estudi de les dades	50	20		1000
Programació	25	20		500
Experimentació	50	20		1000
Elaboració de la memòria del projecte	40	20		800
TOTAL				**3300**
Impostos				**Cost [€]**
21% Sobre el total				**955,5**
COST DEL MODEL				**5505,5**

Impacte ambiental

L'impacte ambiental d'aquest projecte dependrà de la implementació que es doni al model creat. El model te un consum d'energia per ser utilitzat, però l'impacte ambiental real vindria en el moment de la seva implementació a les llotges. Les implicacions a considerar són:

- Reducció de personal degut a l'automatització de processos i la conseqüent reducció de mobilitat
- Consum energètic del model
- Augment de les compres online i la possible reducció en mobilitat que això suposaria
- Augment del consum de gambes locals i el possible impacte que podria tenir sobre l'espècie marina

Conclusions

Revisió dels objectius plantejats

Primer de tot destacar l'assoliment dels requisits previs del treball. Calia obtenir imatges el més semblants possible a les de les llotges que volen implementar l'automatització del *pricing* de les gambes. Aquesta feina no queda reflectida a la memòria del treball però ha estat clau per a l'assoliment dels objectius. Ha estat una tasca especialment complicada pel moment temporal en el que s'ha desenvolupat el treball, en el que la mobilitat ha estat molt restringida. Per tant, es valora molt positivament aquesta primera tasca.

El principal objectiu era la creació d'un model que permetés comptar el nombre de gambes presents a una imatge. Els resultats demostren l'assoliment de l'objectiu. Tot i que el model no és capaç de comptar totes les imatges de qualsevol imatge, sí que compta amb precisió les imatges representatives de les llotges. Com que són aquest tipus d'imatge les que es subministraran al model, podem concloure que l'objectiu s'ha assolit.

Línies d'evolució futures

Una possible via de continuació és la implementació del model a les llotges catalanes. En primer lloc posaria a prova el model, però a més a més seria una oportunitat per fer una aplicació real d'un projecte universitari. La instal·lació del sistema posaria de manifest nous reptes que de ben segur serien molt interessants per a un estudiant.

Agraïments

Primerament agrair al professor Vicenç Parisi Baradad per l'ajuda rebuda al llarg d'aquest projecte. Sobretot per la seva total disponibilitat i ganes d'ajudar per a que avancés de la millor manera possible. Al llarg d'un projecte d'aquestes característiques sempre hi ha bons i mals moments, però en Vicenç sempre ha tingut paraules de motivació i complicitat. Ha estat un plaer treballar amb ell i considero que el seu suport ha estat important per la realització del projecte.

Per últim, donar les gràcies als meus pares, Ester i Lorenzo. Han estat un suport incondicional i han fet tot el possible per que disposés de les millors condicions per fer el treball. Han sabut entendre els requeriments d'aquest projecte i m'han lliurat d'altres responsabilitats

Bibliografia

Referències bibliogràfiques

[1] H. Heindenreich, “What are the types of machine learning? - Towards Data Science,” 2018. https://towardsdatascience.com/what-are-the-types-of-machine-learning-e2b9e5d1756f (accessed Jun. 14, 2020).

[2] D. Pickell, “Structured vs Unstructured Data – What’s the Difference?,” 2018. https://learn.g2.com/structured-vs-unstructured-data (accessed Jun. 14, 2020).

[3] J. Brownlee, “Difference Between Algorithm and Model in Machine Learning,” 2020. https://machinelearningmastery.com/difference-between-algorithm-and-model-in-machine-learning/ (accessed Jun. 13, 2020).

[4] T. Shin, “Machine Learning Models - Towards Data Science,” 2020. https://towardsdatascience.com/all-machine-learning-models-explained-in-6-minutes-9fe30ff6776a (accessed Jun. 14, 2020).

[5] M. A. Nielsen, “Neural Networks and Deep Learning.” Determination Press, 2015.

[6] J. Jordan, “Normalizing your data (specifically, input and batch normalization).,” Jan. 26, 2018. https://www.jeremyjordan.me/batch-normalization/ (accessed Jun. 16, 2020).

[7] P. Brahmbhatt, “The YOLO Object Detection,” Feb. 24, 2019. https://medium.com/@prashant.brahmbhatt32/the-yolo-object-detection-c195994b53aa (accessed Jun. 16, 2020).

[8] C. Baskin, “Image convolution with an input image of size 7 × 7 and a filter kernel... | Download Scientific Diagram,” Jul. 2017. https://www.researchgate.net/figure/Image-convolution-with-an-input-image-of-size-7-7-and-a-filter-kernel-of-size-3-3_fig1_318849314 (accessed Jun. 17, 2020).

[9] M. Rizwan, “Convolutional Neural Network - In a Nut Shell - engMRK,” Sep. 17, 2018. https://engmrk.com/convolutional-neural-network-3/ (accessed Jun. 17, 2020).

[10] M. D. Zeiler and R. Fergus, “Visualizing and Understanding Convolutional Networks,” 2013.

[11] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, “OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks,” *2nd Int. Conf. Learn. Represent. ICLR 2014 - Conf. Track Proc.*, Dec. 2013, Accessed: Jun. 01, 2020. [Online]. Available: http://arxiv.org/abs/1312.6229.

[12] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection,” 2016. Accessed: Jun. 01, 2020. [Online]. Available: http://pjreddie.com/yolo/.

[13] Tzutalin, “LabelImg, git code,” 2015. https://github.com/tzutalin/labelImg (accessed Jun. 23, 2020).

[14] “Project Jupyter,” 2020. https://jupyter.org/ (accessed Jun. 23, 2020).

[15] “Colaboratory – Google,” 2020. https://research.google.com/colaboratory/faq.html (accessed Jun. 23, 2020).

[16] “About Python™ | Python.org,” 2020. https://www.python.org/about/ (accessed Jun. 23, 2020).

[17] “Why TensorFlow,” 2020. https://www.tensorflow.org/ (accessed Jun. 23, 2020).

[18] “Keras: the Python deep learning API,” 2020. https://keras.io/ (accessed Jun. 23, 2020).

[19] A. Karazhay, “Learn to Augment Images and Multiple Bounding Boxes for Deep Learning in 4 Steps,” 2019. https://medium.com/@a.karazhay/guide-augment-images-and-multiple-bounding-boxes-for-deep-learning-in-4-steps-with-the-notebook-9b263e414dac (accessed Jun. 09, 2020).

[20] M. Liang and X. Hu, “Recurrent Convolutional Neural Network for Object Recognition,” 2015.

[21] R. Girshick, “Fast R-CNN,” 2015. Accessed: Jun. 02, 2020. [Online]. Available: https://github.com/rbgirshick/.

[22] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,” 2016. Accessed: Jun. 02, 2020. [Online]. Available: https://github.com/.

[23] C. Szegedy *et al.*, “Going deeper with convolutions,” 2014.

[24] C. Szegedy, V. Vanhoucke, S. Ioffe, and J. Shlens, “Rethinking the Inception Architecture for Computer Vision,” 2015.