Treball de Fi de Màster

Màster Universitari en Enginyeria d'Organització

Xarxes neuronals convolucionals aplicades a la identificació i mesura automàtica

MEMÒRIA

Autor: Guillem Manresa Cid
Director: Vicenç Parisi Baradad

Convocatòria: Setembre 2020

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

Resum

Aquest projecte consisteix en l'entrenament d'una xarxa neuronal convolucional basada en l'arquitectura U-Net, amb l'objectiu d'automatitzar el reconeixement d'exemplars de *aristeus antennatus* per a la seva posterior mesura.

Aquest projecte està emmarcat dins un projecte de recerca o col·laboració entre científics i pescadors, on es pretén estimar de manera automàtica la talla de certes espècies de productes pesquers.

Al llarg d'aquest projecte es construeix una base de dades, fins ara inexistents, amb diferents imatges de l'espècie a identificar i de l'entorn on es durà a terme aquesta detecció. La xarxa neuronal entrenada es basa en una adaptació de la proposada per Kagel en el *Carvana Image Masking Challenge*. Posteriorment es crea un script que identifica els exemplars i estableix una classificació segons la seva mida.

Paraules clau: CNN, xarxes neuronals, *Deep learning*, Arquitectura U-net, *aristeus antennatus.*

Resumen

Este proyecto consiste en el entrenamiento de una red neuronal convolucional basada en la arquitectura U-Net, con el objetivo de automatizar el reconocimiento de ejemplares de *aristeus antennatus* para su posterior medida.

Este proyecto está enmarcado dentro de un proyecto de investigación o colaboración entre científicos y pescadores donde se pretende estimar de manera automática la talla de ciertas especies de productos pesqueros.

A lo largo del proyecto se construye una base de datos, inexistente hasta la fecha, con diferentes imágenes de la especie a identificar, así como del entorno donde se llevará a cabo la detección. La red neuronal entrenada se basa en una adaptación de la propuesta por Kagel en el *Carvana Image Masking Challenge*. Posteriormente se crea un script que identifica los ejemplares y establece una clasificación según su medida.

Palabras clave: CNN, redes neuronales, *Deep learning*, Arquitectura u-Net, *aristeus antennatus.*

Abstract

This project consists on the training of a convolutional neural network based on the U-Net architecture. It aims to automatize the recognition of *aristeus antennatus* specimens for its later measure.

This project is framed within a project of investigation or collaboration between scientists and fishers which objective is to automatically estimate the size of certain species of fish products.

During the project, nonexistent so far data base is built with different images of the specie to identify, so as the environment where the identification will be made. The neural network its based in the adaptation of the network proposed by Kagel in the Carvana Image Masking Challenge. Finally, it is developed a script that identifies the specimens and establishes a size classification.

Keywords: CNN, neural network, Deep learning, U-Net Architecture, *aristeus antennatus.*

Agraïments

En aquest apartat m'agradaria agrair a diferents persones la seva ajuda i aportació a l'hora de realitzar aquest projecte.

Al meu tutor, Vicenç Parisi. Gràcies per donar-me l'oportunitat de dur a terme aquest projecte, així com d'ajudar-me en tot el procés. Gràcies per la teva empenta i pro activitat en els moments més difícils i en general, per haver estat sempre disponible i disposat a ajudar-me. A títol personal, crec que la figura d'un tutor participatiu esta infravalorada i desgraciadament no tothom hi posa les ganes i energia que tu l'hi has posat.

Als meus companys Javier Francés, Llorenç Piera, Marc Campmany, Joan Heredia i Joaquim Celma, que en diferent mesura han col·laborat amb el seu coneixement, suggeriments i experiència prèvia, així com amb el seu suport.

A tots vosaltres, moltes gràcies.

Sumari

1. Prefaci

1.1. Origen del projecte

La Intel·ligència Artificial (IA) no és un concepte nou, el terme va ser definit al 1956 per en John McCarthy, un professor d'Stanford. Aquest concepte es pot definir com a intel·ligència humana duta a terme per a màquines; sistemes que aproximen, repliquen, automatitzen, i eventualment, milloren l'actuació dels humans.

No ha sigut fins aquests últims anys que la IA s'ha començat a aplicar a la resolució de problemes concrets. A la figura X es presenten diferents aplicacions basades en IA que es fan servir actualment.

Problem Type	Inputs	Hidden Layers	Output
Image Recognition	Picture(s)	Person? Face? Gender? Age? Hair & eye color?	Is it you? (%)
Loan Approval	Loan application	Income? Credit history? Employment? Marital status?	Will you repay? (%)
Online Ad Placement	Social media profile, browsing history	Demographics? Browsing history metadata	Will you click? (%)

Figura 1. Exemples d'aplicacions de la IA.

L'aparició d'aquestes eines, juntament amb la robotització, han revolucionat el món de la indústria mitjançant el concepte d'indústria 4.0. L'ús de robòtica habilitada amb IA, sistemes de visió per computació, interfícies de conversa, vehicles autònoms i suport en tasques de *forcasting* son exemples de la introducció de la IA al mercat actual.

És aquí, en la visió per computació, on s'origina aquest projecte. En els últims anys, un gran número de publicacions s'ha dedicat a estudiar i millorar les tècniques de reconeixement d'imatge, i poc a poc, es van incorporant al món actual dintre del marc de transformació digital de molts procesos de diferents indústries com a l'inspecció en la fabricació i acoblament de peces, manteniment predictiu, etc.

Aquest projecte apareix de la possible aplicació de la visió per computació a les llotges de peix, on diferents espècies es pesquen, classifiquen i es posen en el mercat cada día. La tasca de classificació d'espècies es tracta d'un procés ideal per a implementar aquest tipus de tecnologia, ja que és un procés mecànic i repetitiu, i és en aquest tipus de processos on les màquines tenen una efectivitat i eficiència més alta que les presones.

1.2. Motivació

La conjuntura actual, en la que cada cop es disposa de més quantitat de dades i de millor qualitat, juntament amb els objectius de millorar l'eficiència i efectivitat dels processos, i en última instància reduir els costos associats a aquests ha propiciat l'auge d'aquesta vesant tecnològica.

Així doncs, la principal motivació és la d'implementar les noves tecnologies, la Intel·ligència Artificial i la visió per computació a les llotges de peix, amb la finalitat de millorar el procés de classificació i recompte d'espècies. En aquest projecte en concret, es treballarà amb una única espècie, la gamba vermella o *aristeus antennatus.*

1.3. Requeriments previs

Dintre dels requeriments previs d'aquest projecte no es recull cap tipus d'experiència prèvia ni coneixement del sector de pescadors ni de l'espècie amb la que es treballarà.

D'altra banda es fa necessari tenir nocions de programació en Python i coneixements en entorns de programació semblants a Jupyter Notebook o Google Colab.

També serà d'ajuda coneixements en *Machine Learning* o IA, tot i que no és imprescindible.

2. Introducció

2.1. Objectius del projecte

L'objectiu d'aquest treball està inclòs en un projecte més gran (la completa implementació del reconeixement, classificació i mesura d'espècies a la llotja de pescadors). Així doncs, en aquest treball es presenten els següents objectius principals:

- Automatització del reconeixement de gambes vermelles.
- Automatització del recompte i de la mesura dels individus reconeguts.

Es pot observar una clara dependència entre ambdós objectius, per tant, el primer objectiu s'ha de complir indispensablement per a poder assolir el segon.

A més a més, per tal d'assolir els objectius principals s'hauran d'assolir diferents fites:

- Implementació d'un algoritme de *data augmentation.*
- Creació d'una base de dades d'entrenament per a l'algoritme.

2.2. Abast del projecte

L'abast d'aquest treball està lligat a la consecució dels objectius anteriorment descrits.

Així doncs, l'abast del projecta comprendrà des de la recopilació d'imatges i la creació de la base de dades fins a una primera aproximació a un mètode de recompte i mesura de l'espècie seleccionada passant per l'entrenament d'una xarxa neuronal encarregada del reconeixement de les gambes en un entorn real.

3. Anàlisi de l'estat inicial

En aquest capítol es presenta una introducció a les xarxes neuronals i a l'arquitectura emprada en aquest projecte. Seguidament es descriuen les eines més importants utilitzades en la realització del projecte.

3.1. Introducció teòrica

3.1.1. Xarxes neuronals

Les xarxes neuronals son la base de la Intel·ligència Artificial. Aquestes xarxes funcionen i han sigut creades imitant el funcionament de les neurones humanes.

Per a entendre el funcionament de les xarxes neuronals cal tenir clar com funcionen els Componentes d'aquestes; les neurones. Una neurona és el component més bàsic d'aquestes xarxes que funciona rebent entrades (dades, imatges, impulsos elèctrics, etc.), processant-ne la informació, i modificant-ne la sortida, que es dirigirà a la següent neurona. D'aquesta manera, connectant neurones entre elles es crea una xarxa neuronal.

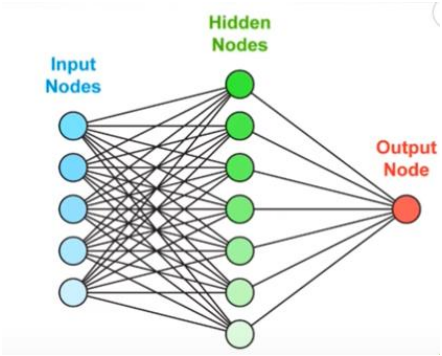


Figura 2.a. Exemple d'una xarxa neuronal.

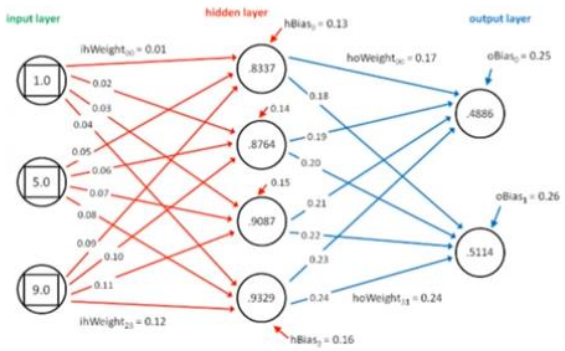


Figura 2.b. Exemple de pesos i connexions d'una xarxa neuronal.

No totes les neurones que formen una xarxa neuronal han de ser exactament igual, i per tant, tampoc ho són les connexions entre elles. Així doncs, en funció de la tipologia de les neurones que formen una xarxa, es pot establir una classificació de xarxes neuronals, amb funcionalitats diferents.

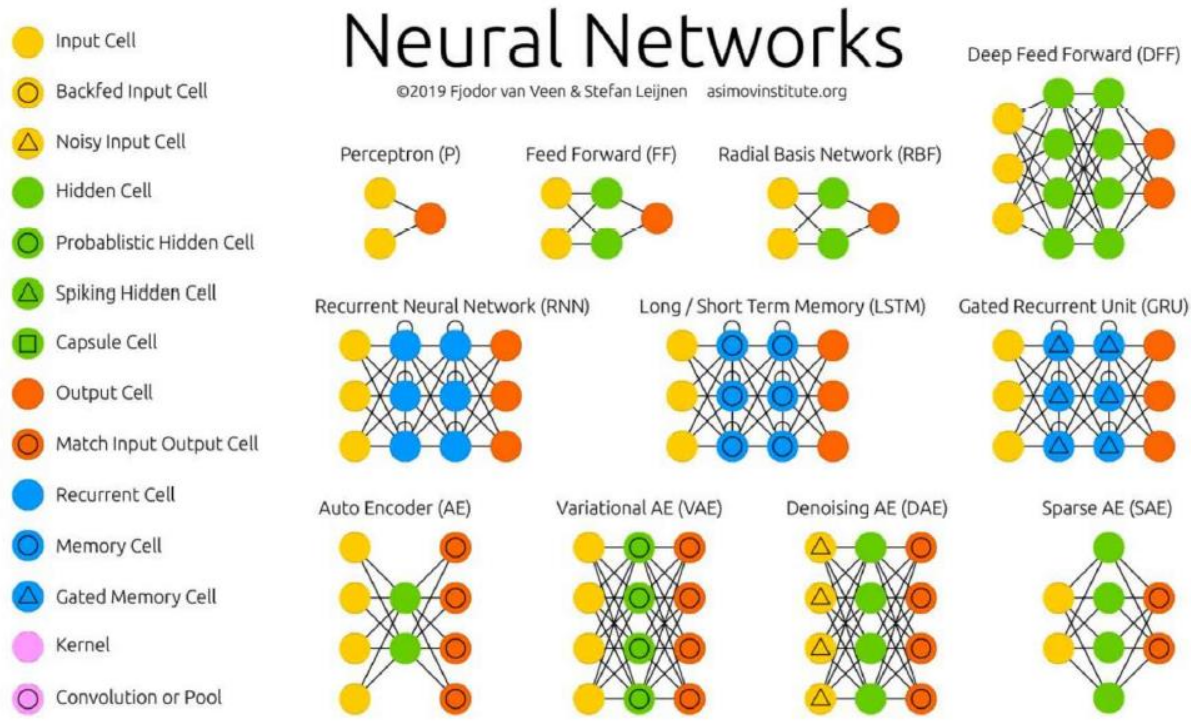


Figura 3. Classificació de les xarxes neuronals.

Una xarxa neuronal pot ser més o menys complexes en funció del número de neurones i capes que la formin. Prenent com a exemple la xarxa mostrada en la figura 2.a, una xarxa formada per les següents capes:

- **Capa d'entrada.** És l'encarregada de rebre l'input al sistema i propagar-lo cap a la següent capa. Cal destacar que una mateixa entrada no incideix per igual en totes les neurones de la capa, permeten obtenir una funció global més complexa.
- **Capa interior.** En aquesta capa s'apliquen diferents transformacions al informació rebuda en funció del pes de cada neurona, reduint o augmentant les dimensions de les dades. El número de capes interiors o *hidden layers* varia en funció de la xarxa neuronal.
- **Capa de sortida.** És la capa encarregada de descodificar la informació i donar la resposta de la xarxa, ja sigui una classificació binaria, una imatge, etc.

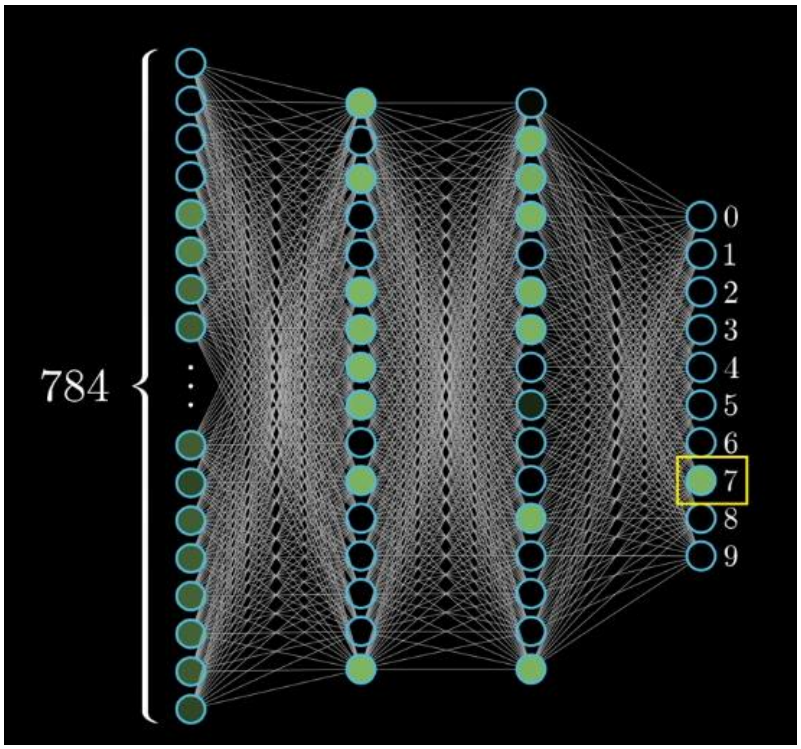


Figura 4. Exemple de la interacció entre les diferents capes d'una xarxa neuronal.

3.1.2. Xarxes neuronals convolucionals

Les xarxes neuronals convolucionals (CNN) són un tipus de xarxa neuronal que funcionen especialment bé per al reconeixement d'imatges, que és en el que es basa aquest projecte.

L'avantatge d'aquest tipus de xarxa respecte a les altres per al reconeixement d'imatges rau en les convolucions i el *pooling* o agrupació. Aquestes operacions aconsegueixen reduir el número de connexions de la xarxa, de manera que s'optimitza tot el procés.

3.1.2.1. Convolucions

Les convolucions son operacions que van processant fragments de la imatge, reduint la dimensió de la imatge d'entrada.

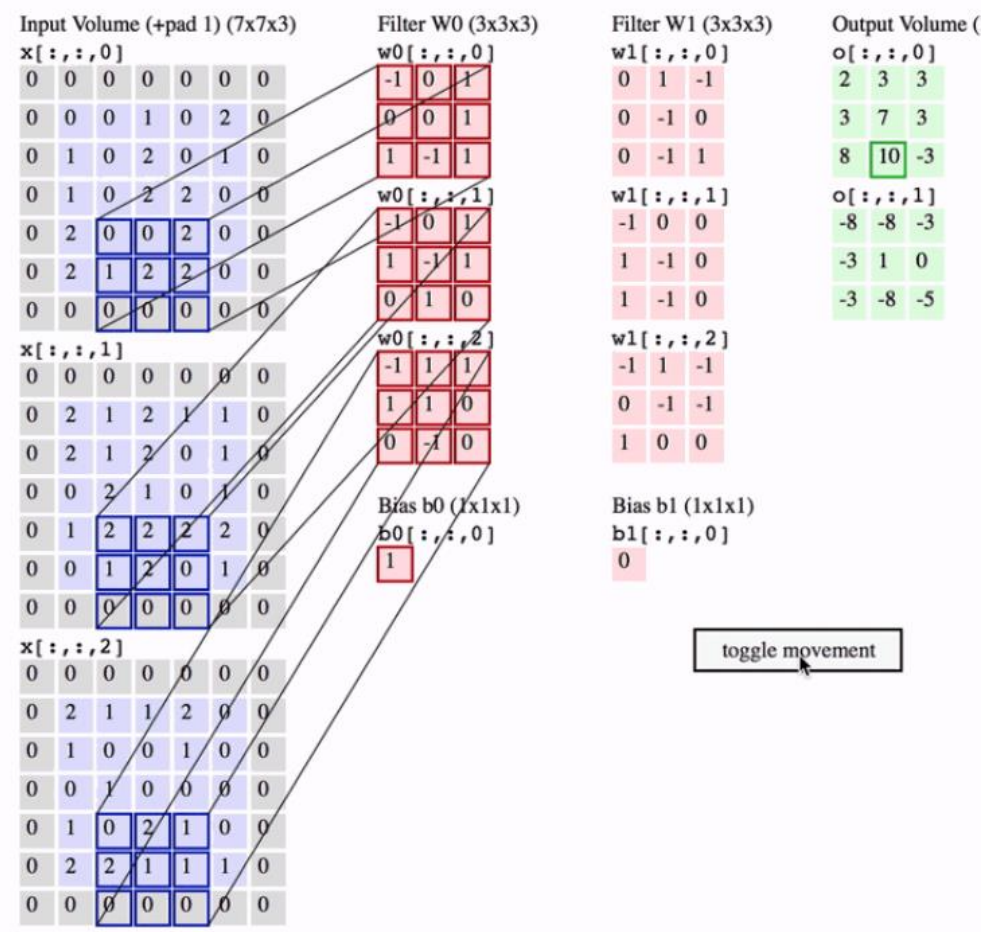


Figura 5. Funcionament d'una convolució.

En una CNN existeixen varies capes de convolucions, on es van aplicant diferents tipus de filtres, cada cop més complexos amb la intenció de trobar patrons en les imatges. D'aquesta manera les primeres convolucions poden estar encarades a reconèixer marges, ombres i contrastos, mentre que les últimes treballen buscant elements més complexes com podrien ser orelles, nassos, cues, etc.

3.1.2.2. Agrupació

Entre cada convolució s'executa una operació d'agrupació, amb els objectius de reduir la mida de la imatge, i per tant les connexions de la xarxa i d'evitar el sobre entrenament.

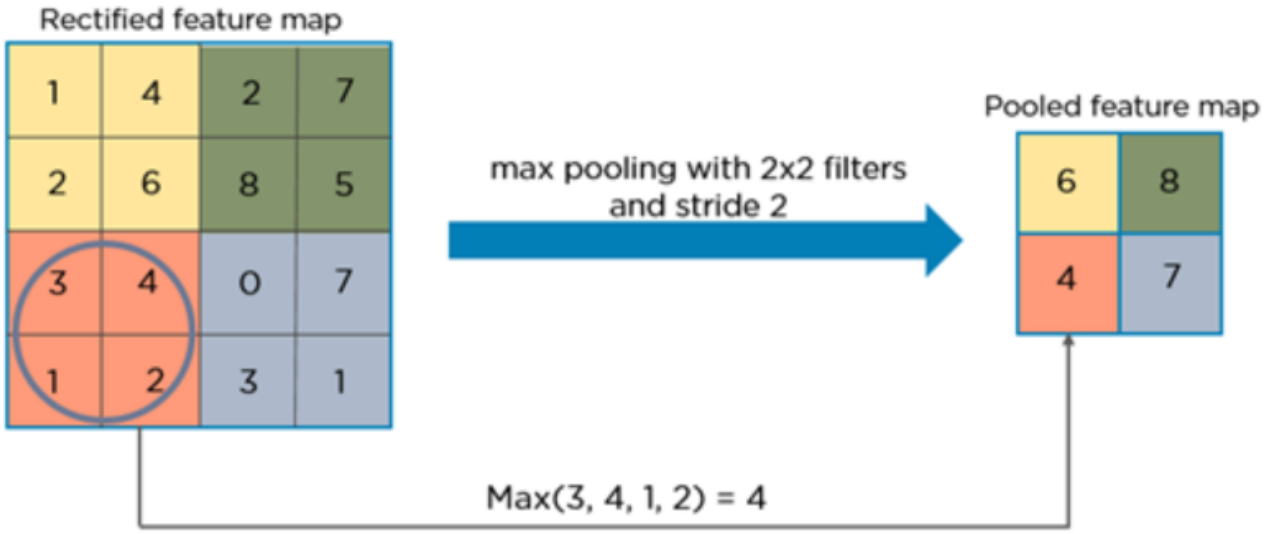


Figura 6. Funionament del procés d'agrupació.

3.1.3. Arquitectura U-Net

L'arquitectura UNET va ser desenvolupada per a la segmentació d'imatges en biomedicina. L'arquitectura es pot dividir en dos grans operacions; *encoding* i *decoding.*

La primera d'elles s'utilitza per a capturar el context en una imatge i consisteix en un conjunt de convolucions i agrupacions, de manera que la mida de la imatge es va reduint, a l'hora que s'augmenta la seva profunditat. L'objectiu final de l'operació és el de determinar quins seran els elements de la imatge que es localitzaran.

La segona operació consisteix en una expansió simètrica que s'utilitza per a localitzar d'una manera precisa els elements identificats en l'operació anterior. Aquesta localització es aconsegueix mitjançant una sèrie de convolucions transposades.

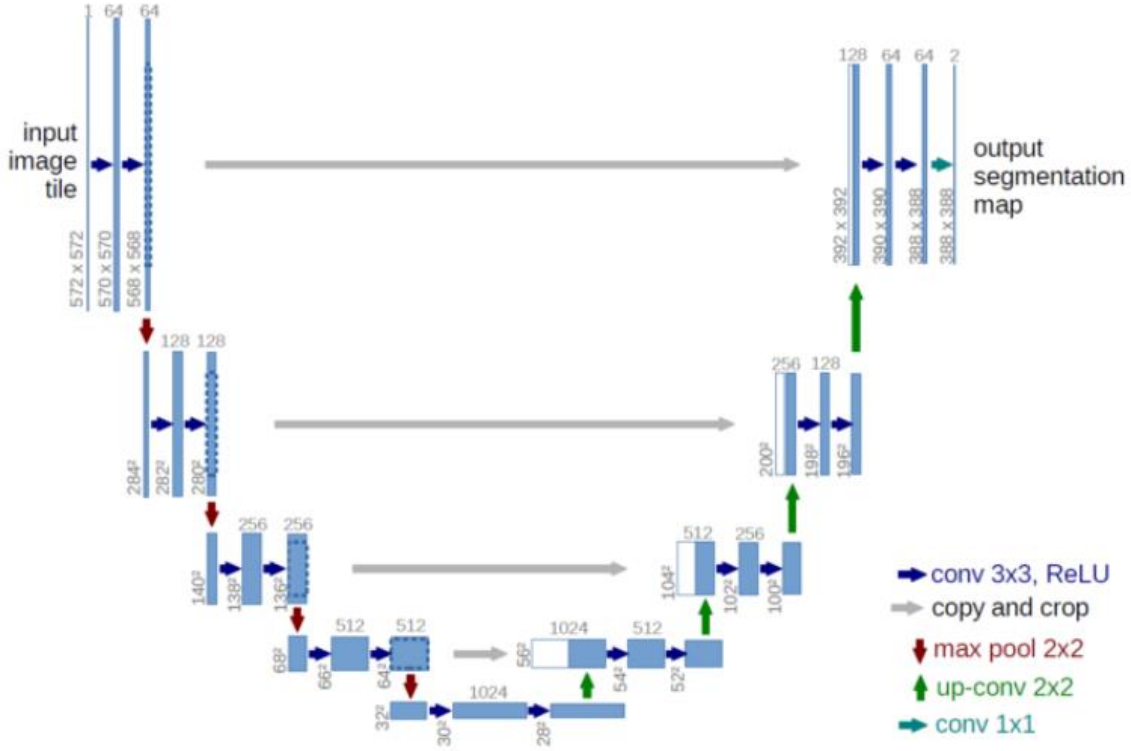


Figura 7. Estructura de l'arquitectura UNET.

Així doncs, es treballa amb una xarxa totalment convolucional (FCN), ja que només conté capes convoluionals, i permet acceptar imatges de qualsevol mida.

El principal avantatge d'aquesta arquitectura, i pel qual s'ha seleccionat per a dur a terme aquest projectes rau en que permet realitzar una segmentació d'imatges de molt bona qualitat amb una quantitat d'imatges d'entrenament baixa, ja que és capaç de convergir ràpidament.

3.2. Eines utilitzades

Al llarg d'aquest projecte s'han utilitzat un conjunt de recursos i llibreries ja existents. Els més transcendents per al projecte es descriuen a continuació.

3.2.1. LabelMe

LabelMe es tracta d'una iniciativa del MIT que persegueix l'objectiu d'oferir una eina d'anotació d'imatges online per tal de construir bases de dades per a la investigació de la visió per computador. En aquest projecte en concret, s'ha utilitzat una adaptació de la iniciativa que permet treballar en local directament des del terminal de l'ordinador. Ha sigut necessària una lleugera modificació dels scripts d'aquesta adaptació per a adequar-la a aquest projecte.

En la figura 8 es mostra l'entorn de treball de l'eina.

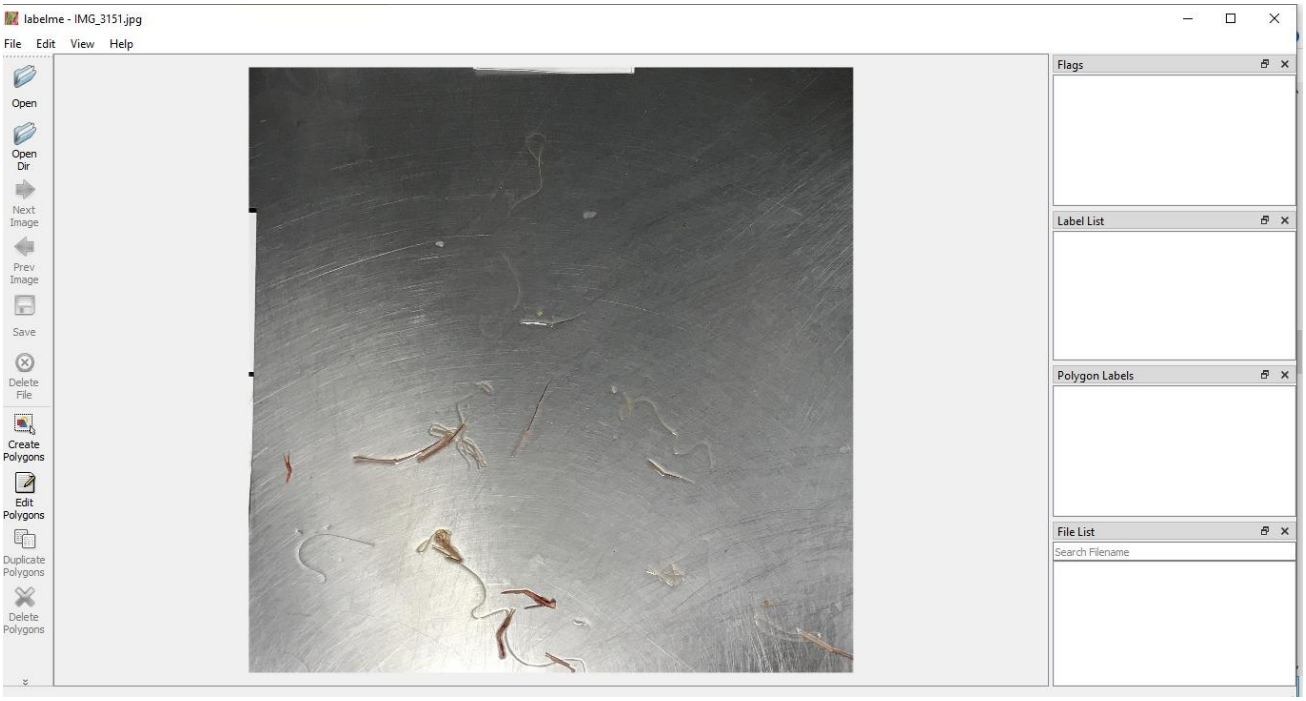


Figura 8. Entorn de treball de l'eina LabelMe.

3.2.2. Jupyter

Project Jupyter és un projecte *open-source* que ha evolucionat com a un entorn interactiu de computació per als *data scientists*. S'ha desenvolupat en obert a GitHub, de manera consensuada per la comunitat de Jupyter.

El notebook està estructurat de manera que es poden executar cel·les independents de codi, en l'ordre desitjat i incloure cel·les de text entre el codi, de manera que es pugui seguir el fil del script de manera senzilla. L'execució dels scripts es realitza en local.

Figura 9. Entorn de treball de l'eina Jupyer Lab.

3.2.3. Google Colaboratory

Google Colaboratory és una plataforma online basada en el projecte Jupyter. A nivell de funcionalitat ofereix pràcticament les mateixes eines que Jupyter Lab, però a diferència d'aquest, la computació es realitza en remot. D'aquesta manera GoogleColab ofereix de manera gratuïta accés a hardware de computació més potent que les màquines comuns, oferint fins a 25 GB de memòria RAM i a unitats gràfiques (GPU) i tensorials (TPU).

Per contrapartida, les dades amb les que es treballa han d'estar emmagatzemades online, en aquest cas s'ha utilitzar Google Drive. Aquest fet limita la capacitat d'emmagatzematge gratuïta.

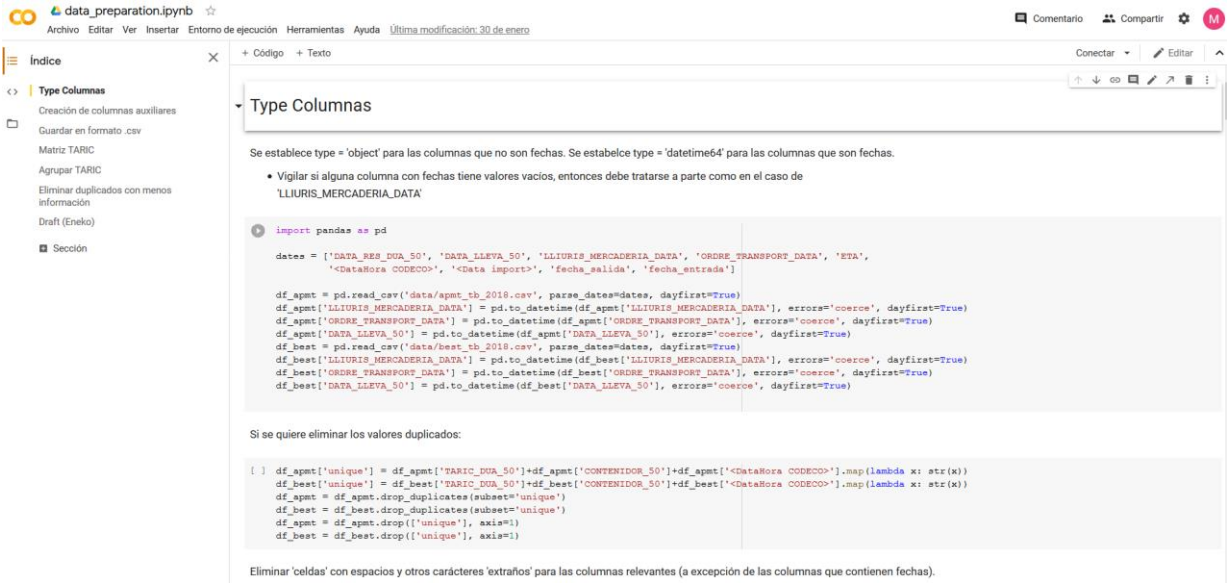


Figura 10. Entorn de treball de l'eina Google Colab.

3.2.4. Llenguatge Python

Python és un llenguatge de programació d'alt nivell que amb la seva sintaxis permet expressar conceptes de manera més senzilla que altres llenguatges. Es tracta d'un llenguatge que evoluciona, de codi obert i que suporta diversos paradigmes com la programació orientada a objectes, imperativa, funcional o procedimental.

Es tracta del llenguatge líder en l'àmbit de la *data science*, ja que es disposa de moltíssimes llibreries gratuïtes encarades a aquest tipus de problemes.

3.2.5. Llibreria TensorFlow

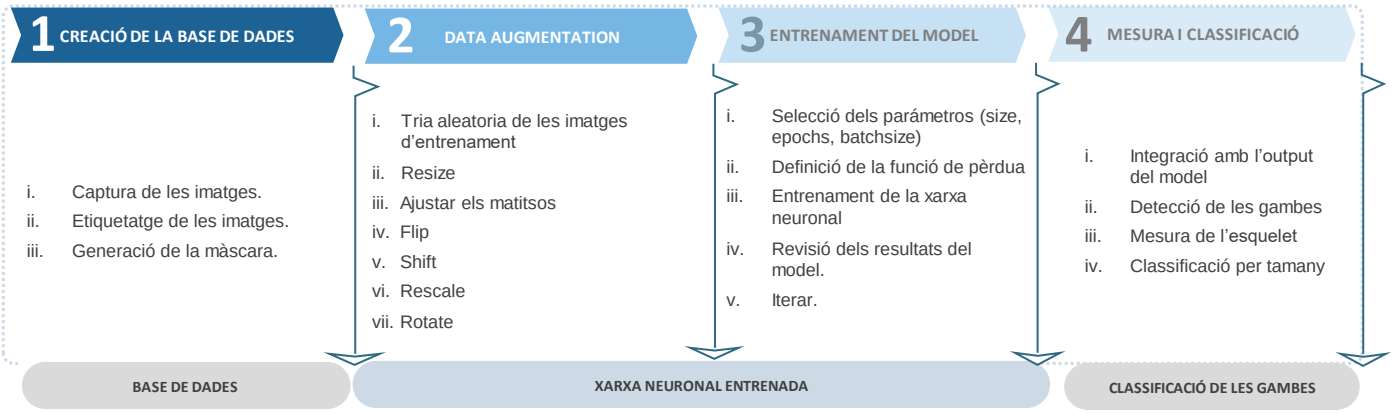
TensorFlow és una llibreria d'ús gratuït especialitzada en *Machine Learning* que persegueix l'objectiu de dissenyar i entrenar xarxes neuronals amb facilitat.

3.2.6. Llibreria Keras

Keras és una llibreria d'ús gratuït per a xarxes neuronals basada en TensorFlow i Python. El seu principal objectiu és el de reduir la complexitat de TensorFlow, permetent un desenvolupament més senzill i agradable.

4. Metodologia

A continuació es presenta la metodologia que es seguirà en aquest projecte:



Com es pot observar en el quadre metodològic, el present projecte es pot dividir en quatre grans blocs:

- La creació de la base de dades
- El procés de *data augmentation*
- L'entrenament de la xarxa neuronal
- La mesura i classificació dels exemplars

4.1. Creació de la base de dades

En aquest bloc, amb l'objectiu de construir una base de dades per a entrenar la xarxa neuronal, es duran a terme tres activitats: la captura de les imatges, l'etiquetatge de les imatges i la generació de les màscares.

4.1.1. Captura d'imatges

L'objectiu d'aquesta tasca es la recol·lecció i presa de les imatges que constituiran la base de dades utilitzada per a l'entrenament de la CNN.

Degut a la situació excepcional ocasionada per la pandèmica provocada per la COVID-19, aquesta activitat s'ha vist afectada, de tal manera que ha arribat a aturar la captura d'imatges en un entorn real (la llotja de Palamós) fins al final de l'estat d'alarma declarat a Espanya. Així doncs, els primers conjunts d'imatges s'han obtingut de fonts alternatives. Atenent a aquestes circumstàncies, es pot establir una classificació de les imatges emprades en aquest projecte en funció de la seva obtenció.

- **Imatges obtingudes de la xarxa.** Inicialment, per tal de poder començar a treballar i a

familiaritzar-se amb les eines, i degut a les restriccions de mobilitat es va començar a treballar amb imatges trobades a la xarxa. Aquestes imatges presentaven diferents característiques entre elles.

Figura 11. Exemple d'imatge de la xarxa.

- **Imatges preses en un entorn simulat.** A mesura que les restriccions de mobilitat es van veure alleujades, es va construir un entorn intentant simular les condicions en les que es treballaria a la llotja de pescadors. D'aquesta manera, es van comprar uns quants exemplars de gamba i es van capturar un seguit d'imatges, totes sobre el mateix fons.

Figura 12. Exemple d'imatge en un entorn simulat.

- **Imatges preses en un entorn real.** Finalment, un cop la situació ho va permetre es va procedir a la captura i recepció d'imatges provinents de la llotja de pescadors. Aquestes imatges es van prendre en l'entorn en el que haurà de treballar el model, és a dir, en l'entorn real d'operació. Cal destacar que el gran gruix de les imatges que constitueixen la base de dades són d'aquesta tipologia. Entre les imatges rebudes existeixen tres tipus d'imatges:
    - Imatges del fons, és a dir, la taula de mostratge buida.
    - Imatges amb un únic exemplar de gamba.
    - Imatges amb múltiples exemplars de gamba.

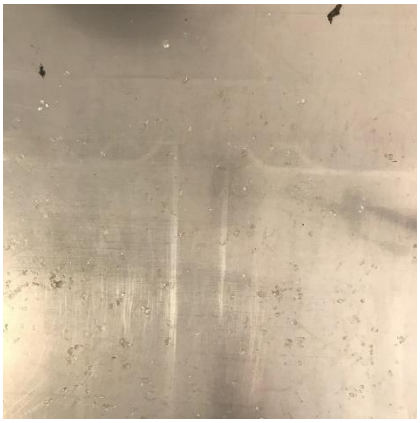


Figura 13. Exemple d'imatge en un entorn real.

Figura 14. Exemple d'imatge en un entorn real..

Figura 15. Exemple d'imatge en un entorn real.

4.1.2. Etiquetatge de les imatges

Un cop en disposició de les imatges que constituiran la base de dades, es fa necessari etiquetar-les per tal de distingir i classificar els diferents objectes que hi apareixen.

En el cas que ocupa aquest projecte, es durà a terme el procés conegut com a segmentació semàntica. Existeixen trens grans tipus de classificació d'imatges en aquest camp, a saber:

- **Segmentació semàntica.** L'objectiu d'aquest tipus de classificació és el d'etiquetar cada píxel d'una imatge amb la classe que li pertany. En el cas que ocupa aquest projecte cada píxel es podrà classificar com a fons o *background* o com a gamba.

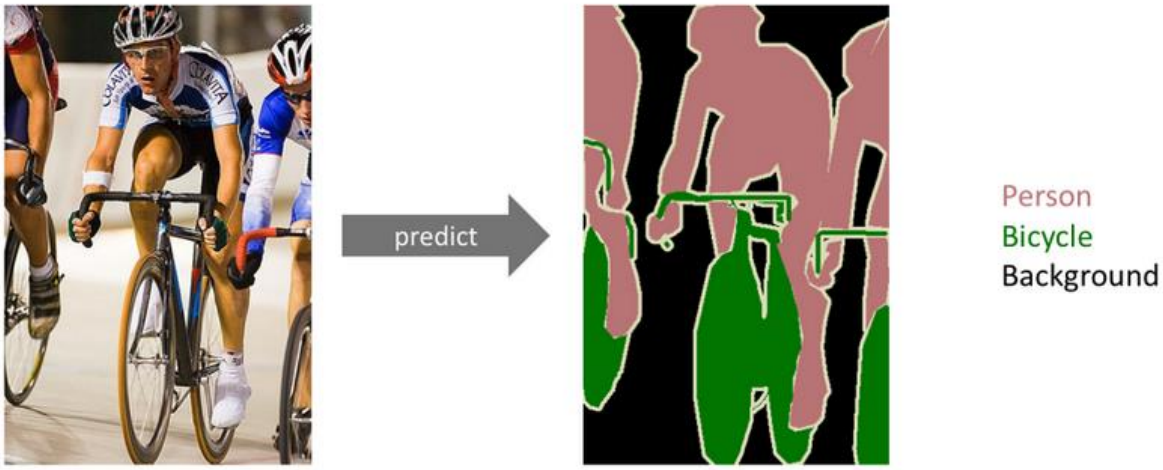


Figura 16. Exemple de segmentació semàntica.

- **Segmentació d'instàncies.** Aquest cas és semblant a l'anterior, però va un pas més enllà. A més a més de classificar cada píxel en funció de la categoria a la que pertany, és capaç de diferenciar entre objectes de la mateixa classe.

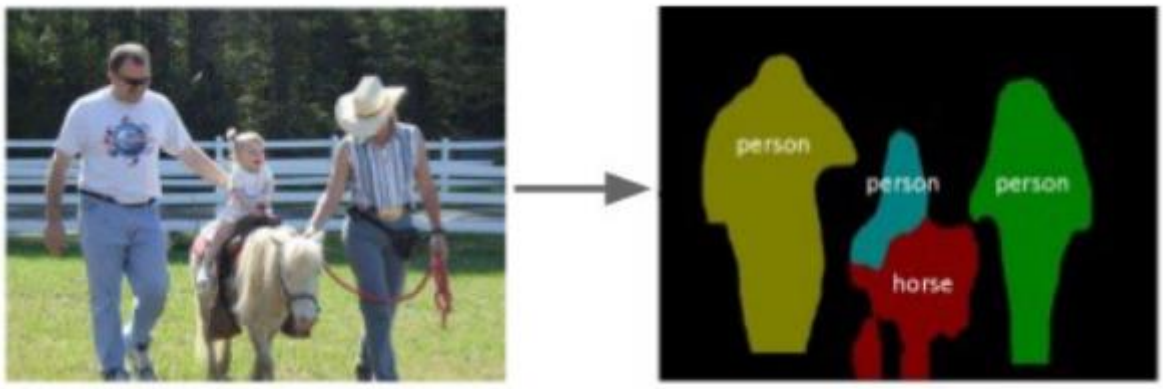


Figura 17. Exemple de segmentació d'instància.

- **Detecció d'objectes.** En aquest mètode es classifica i localitza l'objecte en qüestió, però emmarcant-lo en la seva *bounding box*.

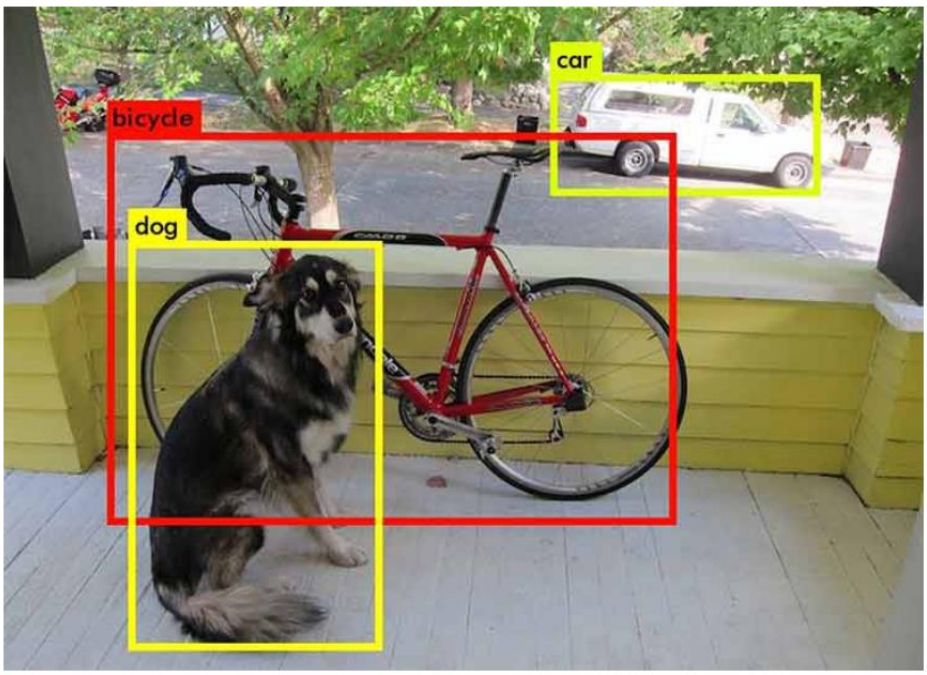


Figura 18. Exemple de detecció d'objectes.

Finalment, amb la intenció d'aclarir les diferències entre els diferents mètodes, a la figura XX es presenta una comparativa entre ells utilitzant la mateixa imatge.

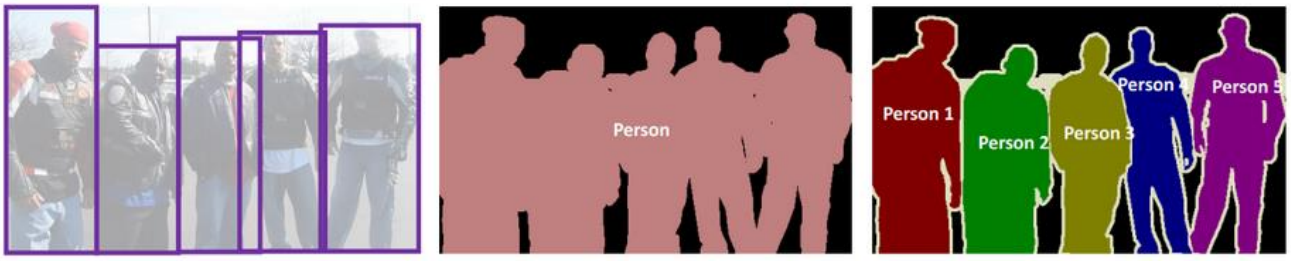


Figura 19. Comparació dels diferents mètodes de classificació.

Un cop determinada l'estratègia de classificació que es seguirà en aquest projecte i les possibles categories que pot prendre un píxel, es procedeix a etiquetar les imatges.

Per tal de dur a terme aquest procés, s'utilitzarà l'eina anteriorment descrita: LabelMe. El procés d'etiquetatge amb aquesta eina consta dels següents passos:

- **Creació dels polígons.** Un cop oberta una imatge amb l'eina, s'han de crear els diferents polígons, punt per punt, per tal de delimitar les zones de la imatge que es vol classificar com a gamba. En la figures 20 i 21 es mostra el procés de creació d'aquests polígons.

1) Es marquen amb el ratolí tots els punts que conformaran el polígon. Les potes dels exemplars s'exclouen.

Figura 20. Definició dels punts que conformen els polígons.

2) Un cop s'han creat tots els polígons es guarda la imatge generant l'arxiu JSON.

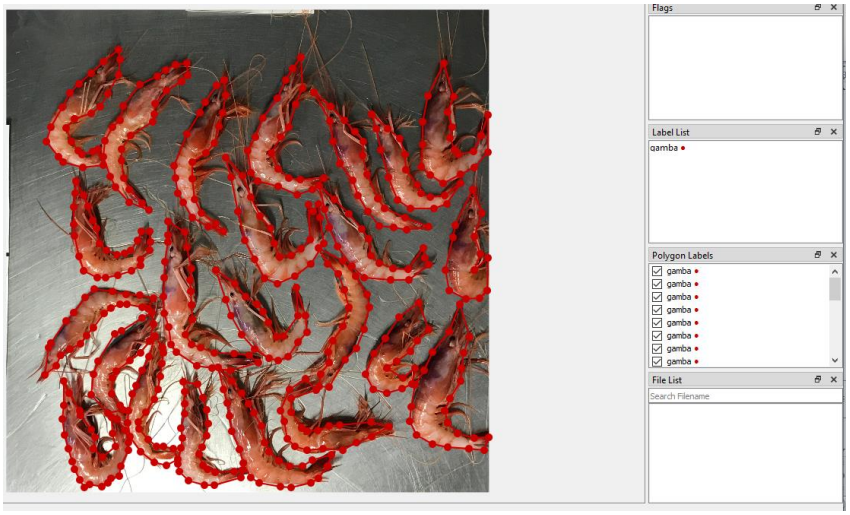


Figura 21. Imatge completament etiquetada.

- **Generació de l'arxiu JSON.** Un cop generat els polígons necessaris, l'eina procedeix a generar un arxiu JSON que conté la següent informació per a cada polígon creat:
  - Etiqueta: en aquest cas, 'gamba'.
  - Punts: la posició de cada punt que conforma el polígon.
  - Amplada i llargada de la imatge

4.1.3. Generació de les màscares

Un cop finalitzat l'etiquetatge, i en disposició dels arxius JSON, es procedeix a la generació de les màscares.

En aquest cas, les màscares consistiran en imatges en blanc i negre on els píxels prendran el color negre si pertanyen a la categoria fons, o blanc si pertanyen a una gamba.

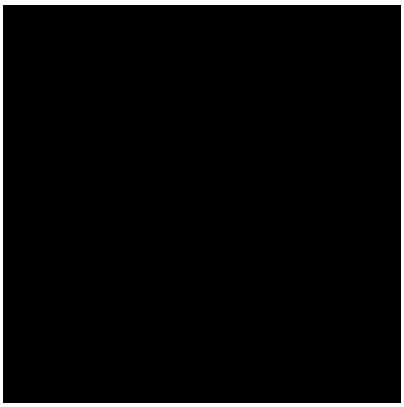


Figura 22. Exemple d'una màscara generada.

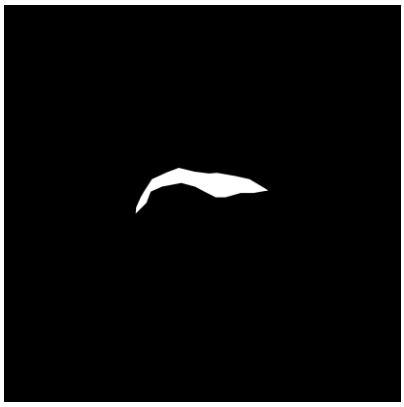


Figura 23. Exemple d'una màscara generada.

Figura 24. Exemple d'una màscara generada.

El procés de generació d'aquestes màscares és molt senzill gràcies a l'adaptació de l'eina LabelMe que s'utilitza, ja que només es fa necessari cridar una comanda des del terminal de la màquina per a que aquesta generi els següents arxius:

- Imatge en extensió PNG.
- Màscara en extensió PNG.
- Arxiu TXT amb les categories: *background* i gamba.
- Imatge amb la màscara superposada en extensió PNG.

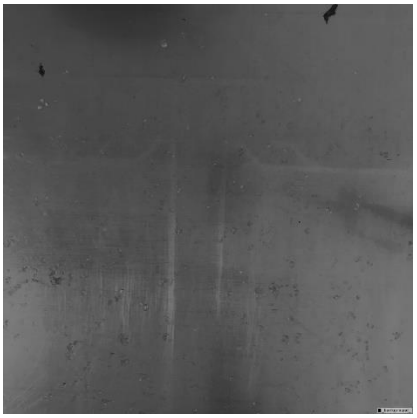


Figura 25. Exemple d'imatge amb la màscara superposada.

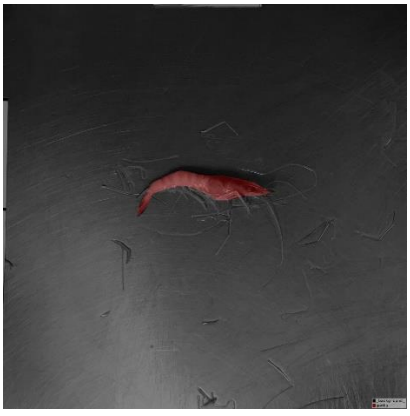


Figura 26. Exemple d'imatge amb la màscara superposada.

Figura 27. Exemple d'imatge amb la màscara superposada.

Un cop realitzat aquest procés per a totes les imatges, s'està en disposició de la base de dades amb la que es duran a terme els experiments. Donat als diferents *timings* en que s'ha estat en disposició de les imatges, els experiments han sigut duts a terme amb diferents subconjunts de la base de dades final.

4.2. Data augmentation

Amb l'objectiu de poder treballar amb una base de dades d'unes certes dimensions, i degut a la complicació i alta inversió de recursos (hores de mà d'obra i memòria de la màquina) requerida en la creació d'una base de dades d'aquestes característiques, en aquest projecte s'ha optat per dur a terme un seguit de transformacions a les imatges contingudes en la base de dades creada en el punt anterior.

Per tal d'optimitzar els recursos de la màquina amb la que s'entrenarà la xarxa neuronal, aquest procés es durà a terme en el mateix entrenament de la xarxa. D'aquesta manera es generen les imatges just abans d'entrenar a l'algoritme i al acabar no es desen a cap lloc, estalviant una gran quantitat de memòria i temps de carrega de les imatges.

Abans de dur a terme les transformacions oportunes però, es separen les imatges d'entrenament en dos conjunts: el conjunt d'entrenament (80%) i el de validació (20%). D'aquesta manera, no es transformaran les imatges de validació, i per tant, s'optimitzaran els recursos.

En aquest projecte s'han definit les següents funcions transformadores:

- ***Resize*.** Es variarà la mida de les imatges. Aquesta transformació és opcional, ja que al treballar amb una arquitectura U-Net (xarxa completament convolucional), aquesta no depèn de la mida de l'input. S'ha de tenir en compte per això, que si totes les imatges no tenen la mateixa mida, el *batch size* haurà de ser igual a 1. Aquesta operació s'aplicarà a la imatge i a la seva màscara.
- **Ajust de matisos.** En aquesta operació s'ajustaran els matisos RGB de la imatge de manera aleatòria. Aquesta transformació només s'aplicarà a la imatge.
- ***Flip.*** Es donarà la volta a la imatge respecte al seu eix central amb una probabilitat del 50%. Aquesta transformació s'aplicarà a la imatge i a la seva màscara.
- ***Shift*.** Es desplaçarà horitzontal o verticalment de manera aleatòria la imatge. Aquesta transformació s'aplicarà a la imatge i a la seva màscara.
- ***Rescale*.** Es durà a terme un re escalat de la imatge per un valor definit. Aquesta transformació s'aplicarà a la imatge i a la seva màscara.
- ***Rotate*.** Es rotarà la imatge per un valor aleatori. Aquesta transformació s'aplicarà a la imatge i a la seva màscara.

4.3. Entrenament del model

En aquest punt es detalla el procés d'entrenament seguit en aquest projecte. Cal destacar que el codi emprat en aquest punt és una adaptació i modificació d'un codi existent [12].

4.3.1. Selecció dels paràmetres

Al llarg del procés d'entrenament, es modificaran diferents paràmetres per tal de millorar el comportament del model. Els paràmetres amb els que es treballarà són els següents:

- ***Batch size.*** Aquest paràmetre determinarà el número d'imatges que entrena la xarxa a cada en cada *step*, de tal manera que si es disposa de X imatges d'entrenament o validació i un *batch size* igual a Y es complirà la relació mostrada en la equació 1.

$$\frac{X}{Y} = steps$$

Equació 1. Relació entre el *batch size* i el número d'*steps*.

  A l'hora de variar aquest paràmetre s'ha de tenir en compte que un valor massa gran pot afectar considerablement als recursos de la màquina utilitzada per a dur a terme l'entrenament. Així doncs, un valor massa alt podria fer que l'entrenament s'aturés degut a la saturació de la memòria GPU. En aquest projecte es treballarà amb *batch sizes* estàtics, i aquests prendran el valor de 4 o 8.
- ***Epochs***. Aquest paràmetre influirà directament sobre la duració de l'entrenament de manera proporcional. Un *epoch* o època fa referència al número de vegades que les dades d'entrenament passaran per la xarxa per a que aquesta aprengui d'ells. Dintre de cada època es duran a terme en número determinat d'iteracions, que serà igual al número d'*steps*.
  Així doncs, aquest paràmetre determinarà en quin punt s'atura l'entrenament, per tant, és tracta d'un paràmetre crític. Si s'estableix un valor massa baix, es pot aturar l'entrenament abans de que el model hagi convergit. D'altra banda, un valor massa gran pot suposar un malbaratament de recursos, ja que a partir de certa època, el model no millorarà.
- ***Image Size*.** Aquest paràmetre indicarà les dimensions de les imatges amb la que es treballarà, i per tant, el número de paràmetres que tindrà el model. La profunditat de les imatges sempre serà la mateixa, ja que es treballarà amb els mateixos canals, RGB. D'altra banda, en aquest projecte es consideren dues mides diferents d'imatges, 256x256 i 512x512. Cal tenir en compte que depenent de la combinació de paràmetres que es defineixi, es pot exhaurir la memòria RAM de la màquina també.

4.3.2. Definició de la funció de pèrdua

La funció de pèrdua serà la que determinarà el rendiment del model, indicant l'error entre el valor estimat i el valor esperat amb la finalitat d'optimitzar els paràmetres de la xarxa neuronal.

En aquest projecte s'utilitzarà una funció especialitzada que combina la entropia creuada binaria i el coeficient *dice*. S'empra aquesta funció ja que és la que millor resultats empírics ha obtingut en aquest tipus de problemes [12].

D'una banda, el *Dice Coefficient* mesura el solapament entre dues mostres. El valor que pren el coeficient varia de 0 fins a 1, on 1 representa un solapament complet. D'altra banda, l'entropia creuada binaria es una variant de l'entropia creuada que s'utilitza en models dels quals l'*output* representa una probabilitat. En aquest cas es fa ús de la binaria ja que el problema que s'afronta es una classificació binaria, on les possibles categories que pot prendre un píxel son: gamba o *background*.

Finalment, l'optimitzador que s'utilitzarà serà l'Adam. Aquest optimitzador tracta de solucionar el problema fixant el rati d'aprenentatge, per fer-ho adapta el rati en funció de la distribució dels paràmetres, augmentant-lo si els paràmetres estan molt dispersos.

4.3.3. Entrenament de la xarxa neuronal

Un cop determinats el valors dels paràmetres a utilitzar es pot procedir a entrenar la xarxa. La xarxa utilitzada en tots els experiments constarà de les mateixes capes i paràmetres a entrenar, ja que no es modificarà l'estructura d'aquesta. Així doncs la xarxa constarà de 75 capes, amb un total de 49.947.585 paràmetres, dels quals s'entrenaran 49.927.873.

Durant el procés d'entrenament es buscarà minimitzar el valor de la funció de pèrdua definida anteriorment fins a arribar al número d'èpoques especificades.

4.3.4. Revisió dels resultats del model

Finalment, un cop entrenat el model, s'analitzaran els resultats obtinguts amb l'objectiu de determinar si es necessari variar algun dels paràmetres o modificar les imatges d'entrenament emprades. Es realitzaran varies iteracions per tal d'assolir l'objectiu de detectar automàticament els exemplars de gamba vermella.

4.4. Mesura i classificació

En aquest punt s'identifiquen els diferents exemplars continguts en la màscara predita per la xarxa neuronal entrenada. Un cop identificats els diferents exemplars continguts en la màscara, es genera el seu esquelet i finalment s'estableix la classificació dels exemplars per mida.

4.4.1. Integració amb l'ouptput del model

El primer pas per tal de dur a terme la classificació es la integració de la sortida del model amb l'algoritme d'identificació d'exemplars. El principal conflicte en aquest punt radica en que el resultat de la predicció és una imatge amb uns píxels de valors diferents a 0 i 1.

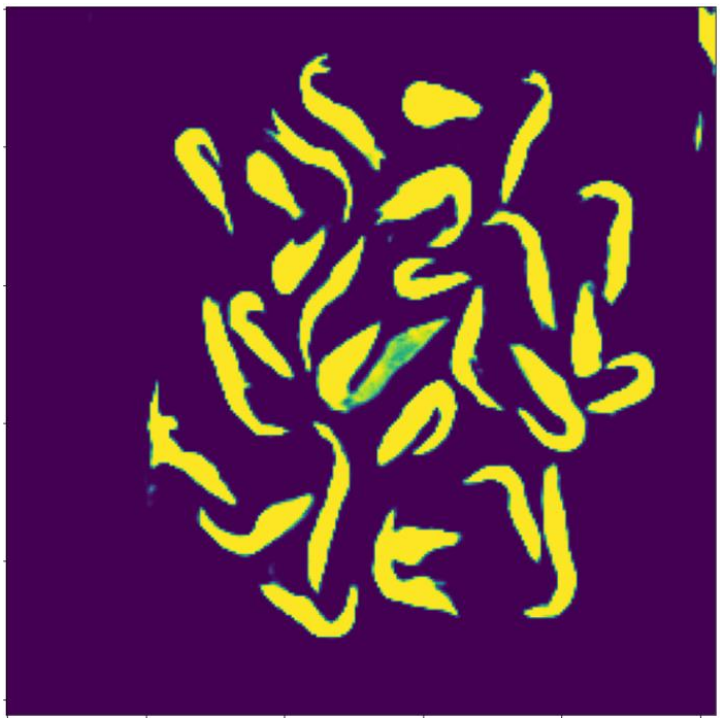


Figura 28. Exemple d'una màscara predita per la xarxa neuronal.

Com es pot observar en la figura 28, en la màscara predita apareixen diferents colors. Tot i semblar que el fons és del mateix color, al examinar en detall els píxels que el conformen, aquests prenen diferents valors, per tant no es pot passar tots els píxels de fons al valor 0 (negre) i la resta a 1 (blanc). Mitjançant un seguit de funciones s'aconsegueix transformar la màscara predita en una imatge en blanc i negre, de manera que estigui llesta per a ser tractada per l'algoritme de detecció.

4.4.2. Detecció de les gambes

Un cop s'està en possessió de la màscara predita en blanc i negre, el primer pas es detectar els diferents exemplars continguts en aquesta màscara. Mitjançant diferents llibreries especialitzades per al tractament d'imatges (skimage i skan), s'identifiquen els diferents píxels amb un valor diferent a 0. D'aquesta manera s'identifiquen els diferents píxels que es troben rodejats per píxels amb un valor de 0. A la figura 29 es mostra la detecció dels exemplars de la màscara mostrada en la figura 28.

Figura 29. Detecció d'exemplars.

4.4.3. Mesura de l'esquelet

Amb els diferents exemplars ja identificats, es procedeix a crear l'esquelet de les gambes. Aquest procés es du a terme inscrivint els diferents exemplars en les seves *bounding boxes* com es mostra en el següent exemple.

- Es parteix per exemple, d'aquestes dues gambes.

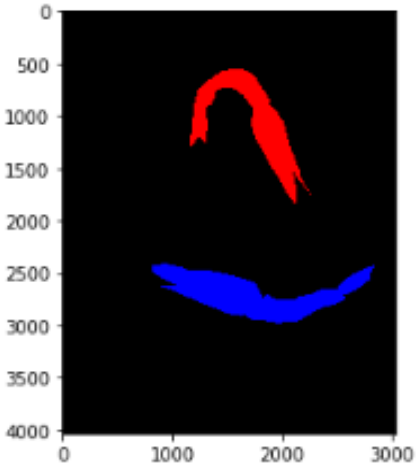


Figura 30. Exemple de la detecció d'exemplars.

- A continuació es troba la *bounding box* de cada exemplar.

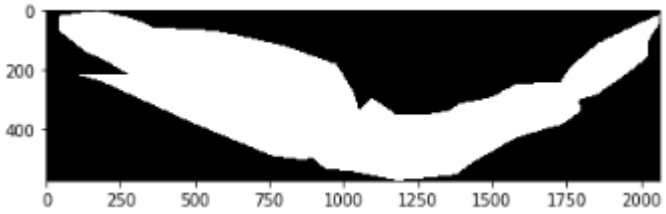


Figura 31. Exemple d'un exemplar en la seva *bounding box.*.

- Posteriorment es defineix el contorn de l'exemplar i es calcula el seu eix mig, trobant els punts equidistants del contorn.

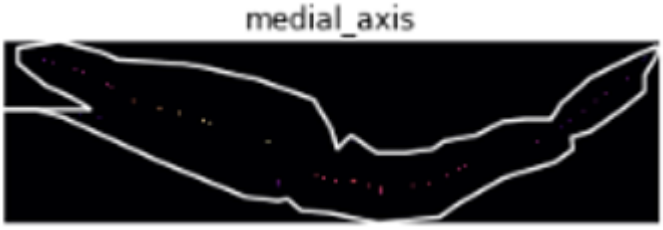


Figura 32. Exemple del contorn i l'ex mig d'un exemplar.

Figura 33. Exemple de l'esquelet d'un exemplar.

- Un cop en disposició de l'eix, es genera l'esquelet de la figura.

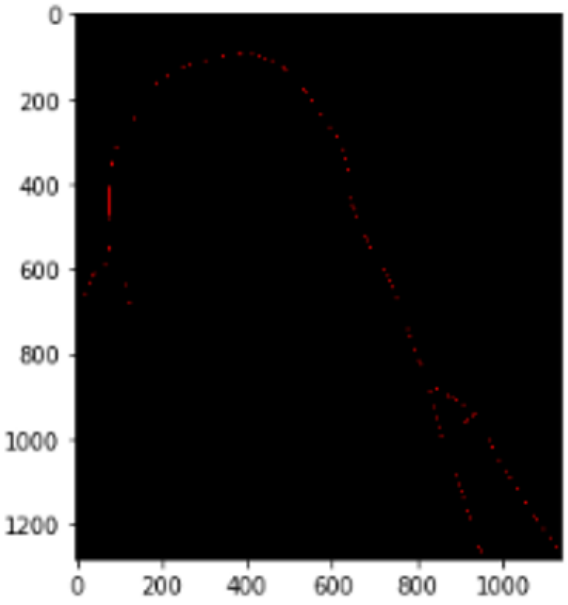


Figura 34. Exemple de l'esquelet d'un exemplar.

Com es pot observar en els esquelets mostrats en les figures 33 i 34, aquests tenen més d'una branca. Per tal de definir la mida de l'exemplar, es recorre el diccionari que conte les diferents branques de l'esquelet i es selecciona la branca més llarga. Aquest procediment es repeteix per a tots els exemplars continguts en la imatge, resultant en un diccionari de Python amb el següent aspecte:

```
{'label_': {'gamba_0': 41.89949493661166,
  'gamba_1': 121.21152392996783,
  'gamba_10': 148.22539674441617,
  'gamba_11': 38.45584412271572,
  'gamba_12': 146.36753236814707,
  'gamba_13': 97.78511705877509,
  'gamba_14': 59.986127185551766,
  'gamba_15': 75.38920874174994,
  'gamba_16': 115.12489168102773,
  'gamba_17': 90.2996207799684,
  'gamba_18': 80.56854249492375,
  'gamba_19': 179.79393923934015,
  'gamba_2': 134.60241217734634,
  'gamba_20': 112.32590180780444,
  'gamba_21': 85.11220722794215,
  'gamba_22': 86.70291724073468,
  'gamba_23': 178.8822509939086,
  'gamba_24': 108.8856221221634,
  'gamba_25': 77.12826280928269,
  'gamba_26': 141.0954544295049,
  'gamba_27': 78.11269837220806,
  'gamba_28': 127.12489168102776,
  'gamba_3': 52.72065437077997,
  'gamba_4': 112.92703777793788,
  'gamba_5': 31.82842712474619,
  'gamba_6': 90.26821566612134,
  'gamba_7': 63.871749307715184,
  'gamba_8': 83.61354418352124,
  'gamba_9': 114.05698009284912}}
```

Figura 35. Exemple d'un diccionari generat amb les mides dels exemplars.

4.4.4. Classificació per mida

Finalment, un cop en disposició de les mides de cada exemplar s'estableix un sistema de classificació que classifica les gambes en tres categories:

- **Grans.** Es consideren grans aquells exemplars amb una mida superior en un 20% a la mitja.
- **Mitjans.** Es consideren mitjans aquells exemplars amb una mida compresa en ± 20% de la mitja.
- **Petits.** Es consideren petits aquells exemplar amb una mida inferior en un 20% a la mitja.

En la figura 36 es mostra un exemple del diccionari tipus que conté la classificació.

```
{'Grans': 8, 'Mitjans': 12, 'Petits': 9}
```

Figura 36. Exemple d'un diccionari amb la classificació dels exemplars.

Arribats a aquest punt es defineix una funció per tal de dur a terme la inferència. Els paràmetres d'entrada de la funció seran el *path* de la imatge que es vulgui passar pel model i el path del model amb el que es vulgui avaluar la imatge. D'aquesta manera, per a la imatge de la figura 37, en la figura 38 es mostra l'output de la funció d'inferència.

Figura 37. Imatge d'entrada a la funció d'inferència.

```
[ ] inferencia(img, model_path)

Requirement already satisfied: skan in /usr/local/lib/python3.6/dist-packages (0.8)
{'Grans': 6, 'Mitjans': 7, 'Petits': 4}
```

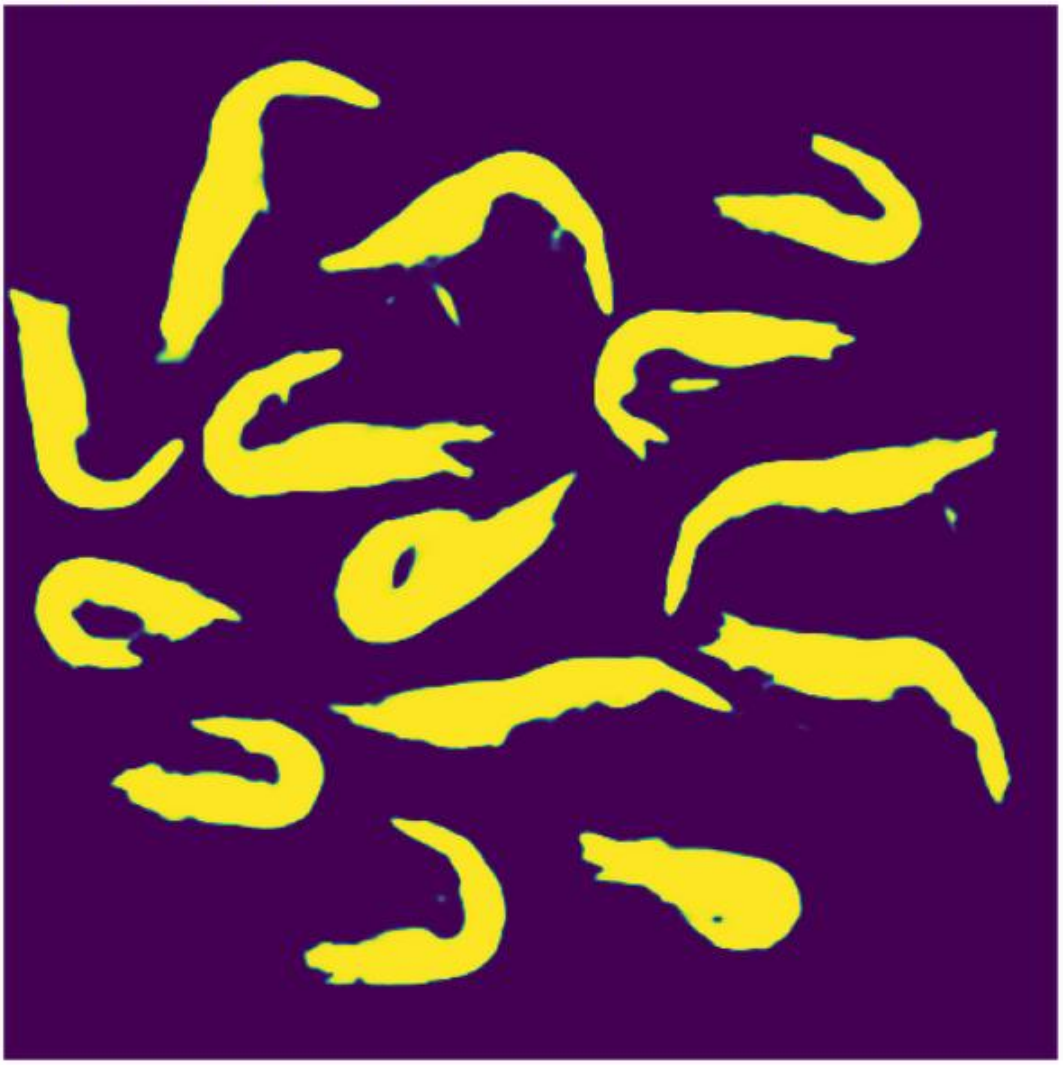


Figura 38. Output de la funció d'inferència.

5. Resultats

En aquest capítol es presenten els resultats dels diferents experiments duts a terme al llarg d'aquest projecte. Tots els experiments presentats seguiran la metodologia desenvolupada en el capítol anterior.

Així doncs, per a cada experiment es presentarà la següent informació:

- Descripció de la base de dades emprada en l'entrenament.
- Exemples de les imatges d'entrenament amb les seves màscares corresponents.
- Valor dels paràmetres utilitzats.
- Resultat de final de l'entrenament del model.
- Gràfics amb l'evolució de la funció objectiu del model.
- Exemples de les imatges de validació amb les seves màscares i les prediccions corresponents.
- Exemples de les imatges de test amb les seves màscares i les prediccions corresponents.
- Observacions en base als resultats obtinguts.

5.1. Experiment 1

Aquest experiment s'ha utilitzat una base de dades formada única i exclusivament per imatges obtingudes de la xarxa i en un entorn simulat. Les dimensions de la base de dades utilitzada són les següents:

- **95 imatges**. Aquestes imatges s'han dividit en 76 imatges de entrenament i 19 de validació.

Per a les imatges de test s'han utilitzat imatges obtingudes en un entorn real, ja que el propòsit de la xarxa es reconèixer les gambes en aquell entorn.

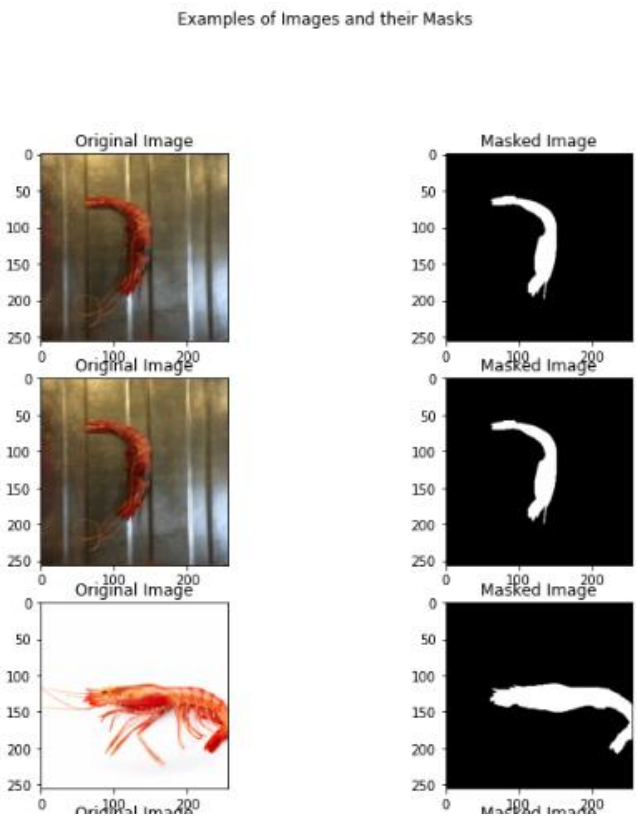


Figura 39. Exemple d'imatges i les seves màscares.

El valor dels paràmetres emprats ha estat el següent:

- Mida de les imatges = (256, 256, 3)
- Batch size = 4
- Epochs = 200

En les figures 40 i 41 respectivament, es mostren els resultats obtinguts en l'entrenament del model amb la base de dades i paràmetres anteriorment descrits, així como l'evolució de la funció de pèrdua.

```
Epoch 200/200
18/19 [===========================>..] - ETA: 0s - loss: 0.1058 - dice_loss: 0.0766
Epoch 00200: val_dice_loss did not improve from 0.10998
19/19 [==============================] - 5s 276ms/step - loss: 0.1056 - dice_loss: 0.0765 - val_loss: 0.2124 - val_dice_loss: 0.1310
```

Figura 40. Últim epoch de l'entrenament.

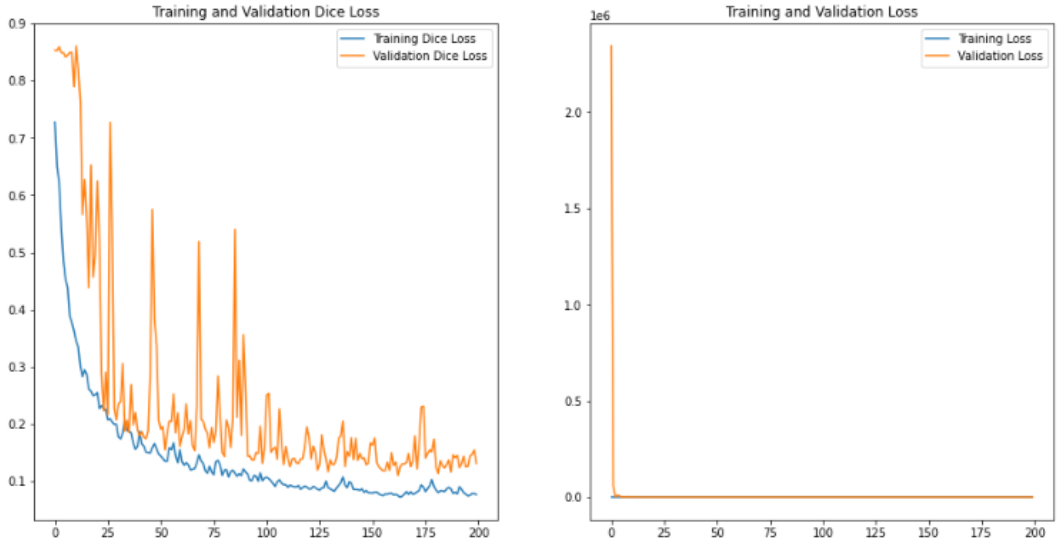


Figura 41. Evolució del valor de la funció de pèrdua.

En les figures 42 i 43 es mostren les prediccions de les màscares fetes per la xarxa de les imatges de validació i test respectivament.

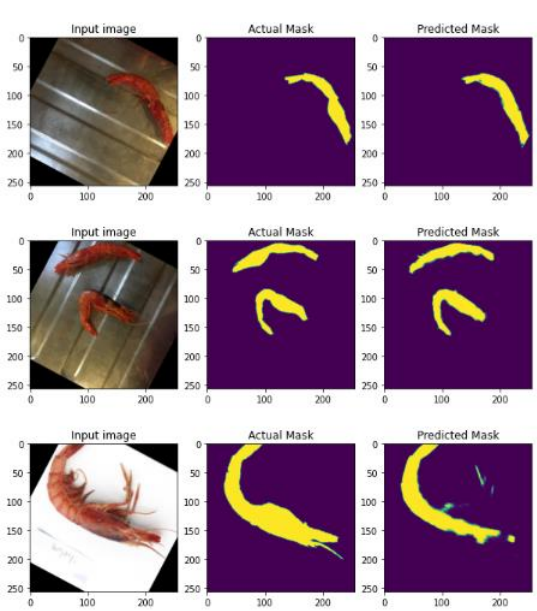


Figura 42. Exemple d'imatges, les seves màscares i la seva predicció.

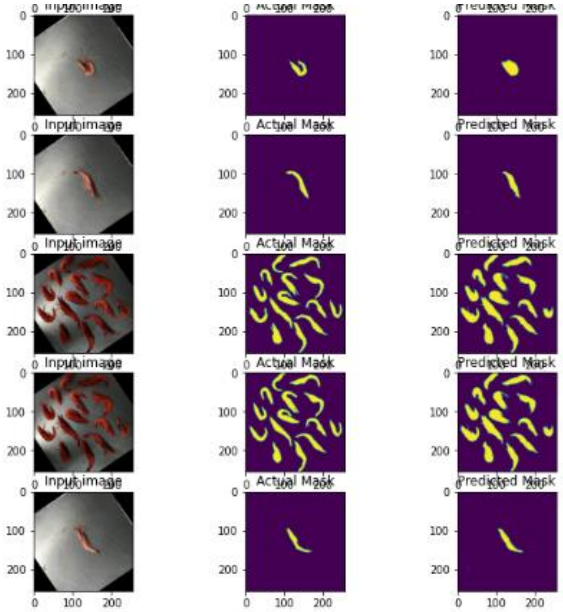


Figura 43. Exemple d'imatges, les seves màscares i la seva predicció.

Com es pot observar, l'evolució de la *validation dice los* oscil·la bastant, el millor valor que pren és de 0,10998, atorgant al model una puntuació de 89,00. Es pot observar que tot i ser una bona aproximació, les prediccions obtingudes no són tot lo acurades que s'espera.

5.2. Experiment 2

Aquest experiment s'ha utilitzat una base de dades formada única i exclusivament per imatges obtingudes en un entorn real. Les dimensions de la base de dades utilitzada són les següents:

- **32 imatges**. Aquestes imatges s'han dividit en 25 imatges de entrenament i 7 de validació.

Per a les imatges de test s'han utilitzat imatges obtingudes en un entorn real, ja que el propòsit de la xarxa es reconèixer les gambes en aquell entorn.

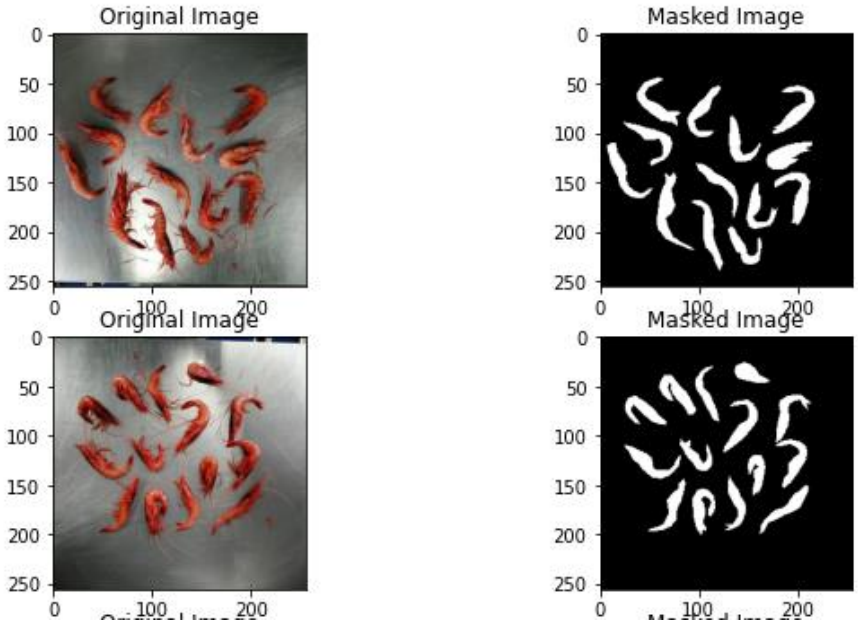


Figura 44. Exemple d'imatges i les seves màscares.

El valor dels paràmetres emprats ha estat el següent:

- Mida de les imatges = (256, 256, 3)
- Batch size = 8
- Epochs = 200

En les figures 45 i 46 respectivament, es mostren els resultats obtinguts en l'entrenament del model amb la base de dades i paràmetres anteriorment descrits, així como l'evolució de la funció de pèrdua.

```
Epoch 200/200
3/4 [=====================>........] - ETA: 0s - loss: 0.0516 - dice_loss: 0.0329
Epoch 00200: val_dice_loss did not improve from 0.09198
4/4 [==============================] - 2s 452ms/step - loss: 0.0571 - dice_loss: 0.0360 - val_loss: 0.2779 - val_dice_loss: 0.0932
```

Figura 45. Últim epoch de l'entrenament.

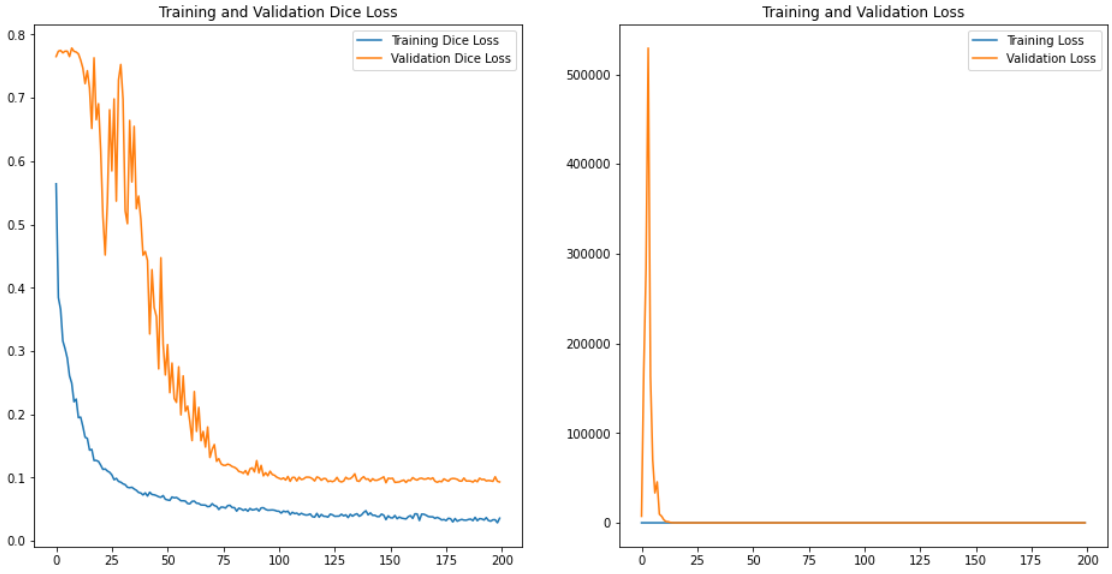


Figura 46. Evolució del valor de la funció de pèrdua.

En les figures 47 i 48 es mostren les prediccions de les màscares fetes per la xarxa de les imatges de validació i test respectivament.

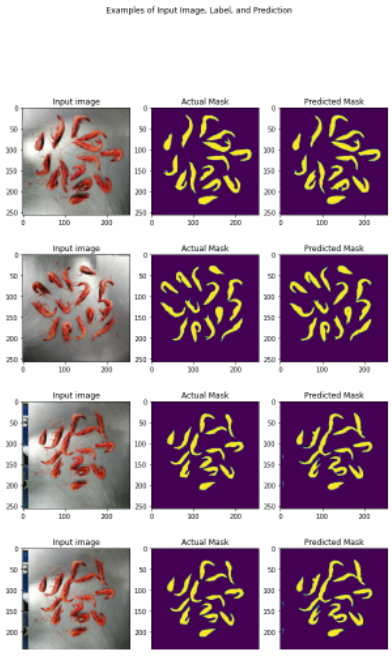


Figura 47. Exemple d'imatges, les seves màscares i la seva predicció.

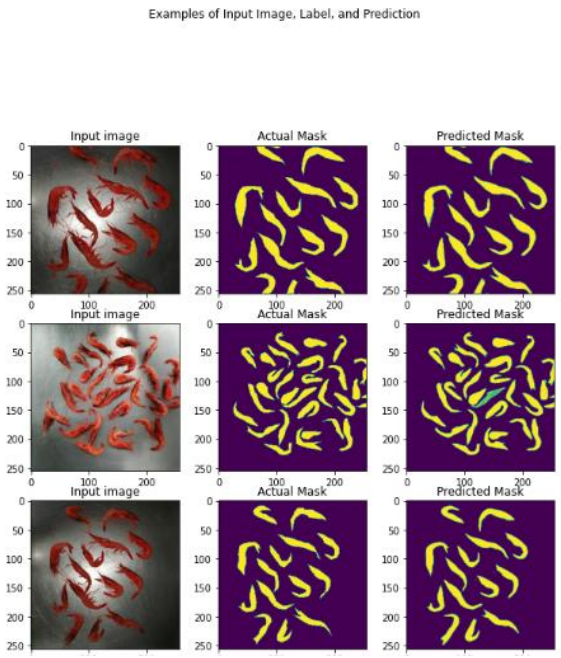


Figura 48. Exemple d'imatges, les seves màscares i la seva predicció.

El millor valor que pren la *validation dice loss* és de 0,09198, atorgant al model una puntuació de 90,80. Es pot observar que tot i treballar casi una tercera part que en l'experiment 1, al treballar amb imatges d'un entorn real, els resultats són millors.

5.3. Experiment 3

Aquest experiment s'ha utilitzat una base de dades formada única i exclusivament per imatges obtingudes en un entorn real. Les dimensions de la base de dades utilitzada són les següents:

- **198 imatges**. Aquestes imatges s'han dividit en 158 imatges de entrenament i 40 de validació.

Per a les imatges de test s'han utilitzat imatges obtingudes en un entorn real, ja que el propòsit de la xarxa es reconèixer les gambes en aquell entorn.

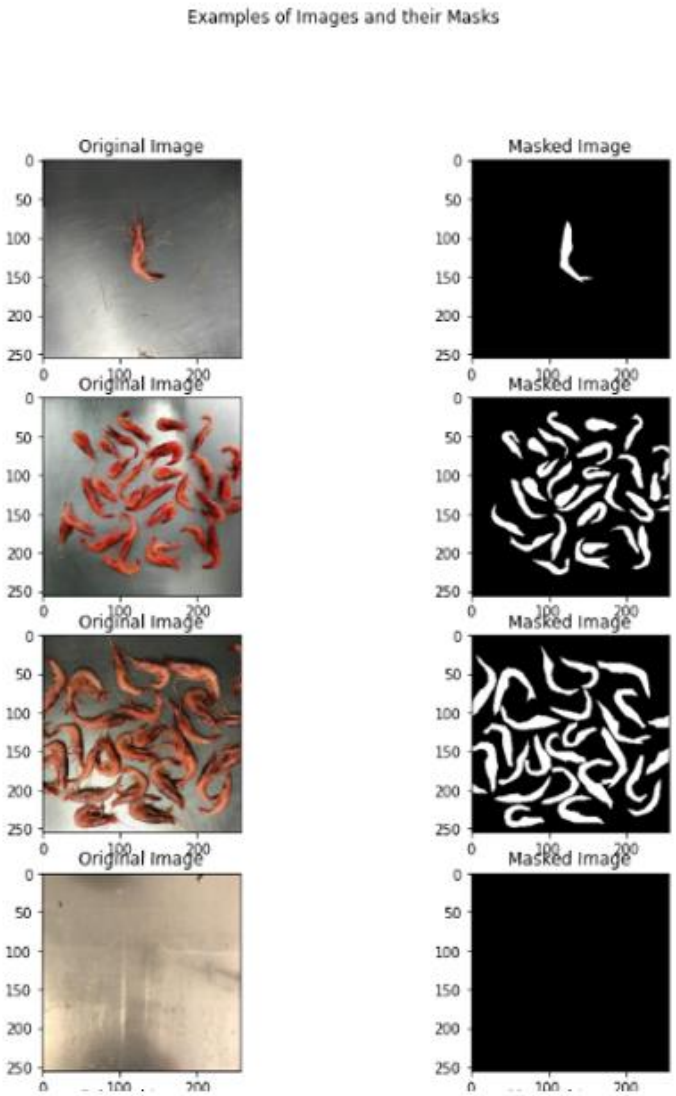


Figura 49. Exemple d'imatges i les seves màscares.

El valor dels paràmetres emprats ha estat el següent:

- Mida de les imatges = (256, 256, 3)
- Batch size = 4
- Epochs = 100

En les figures 50 i 51 respectivament, es mostren els resultats obtinguts en l'entrenament del model amb la base de dades i paràmetres anteriorment descrits, així como l'evolució de la funció de pèrdua.

```
Epoch 100/100
39/40 [============================>.] - ETA: 0s - loss: 0.1351 - dice_loss: 0.1074
Epoch 00100: val_dice_loss did not improve from 0.10950
40/40 [==============================] - 11s 264ms/step - loss: 0.1341 - dice_loss: 0.1069 - val_loss: 0.1359 - val_dice_loss: 0.1106
```

Figura 50. Últim epoch de l'entrenament.

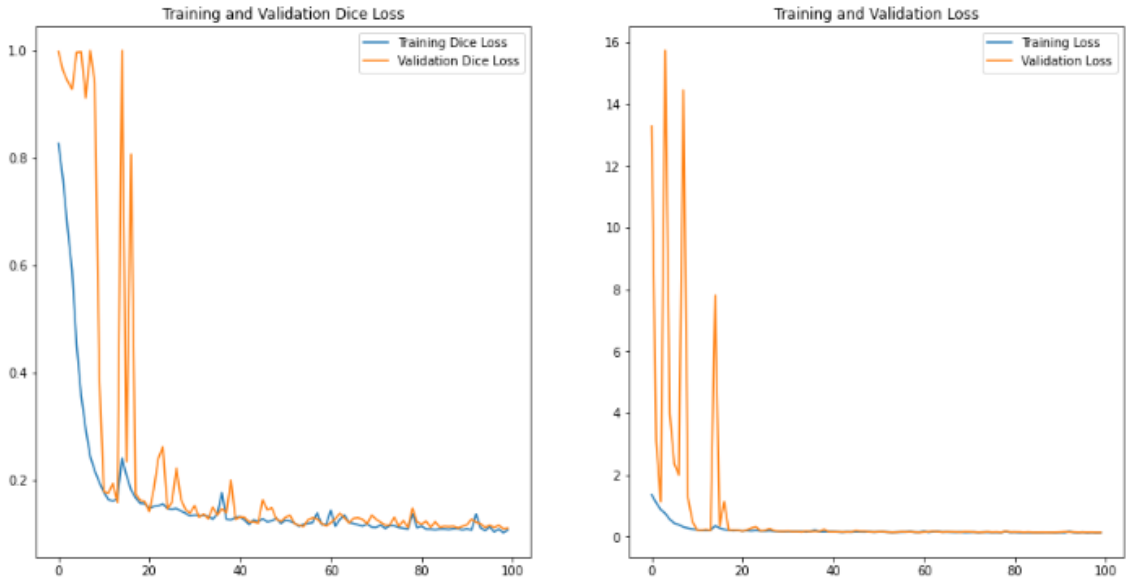


Figura 51. Evolució del valor de la funció de pèrdua.

En les figures 52 i 53 es mostren les prediccions de les màscares fetes per la xarxa de les imatges de validació i test respectivament.

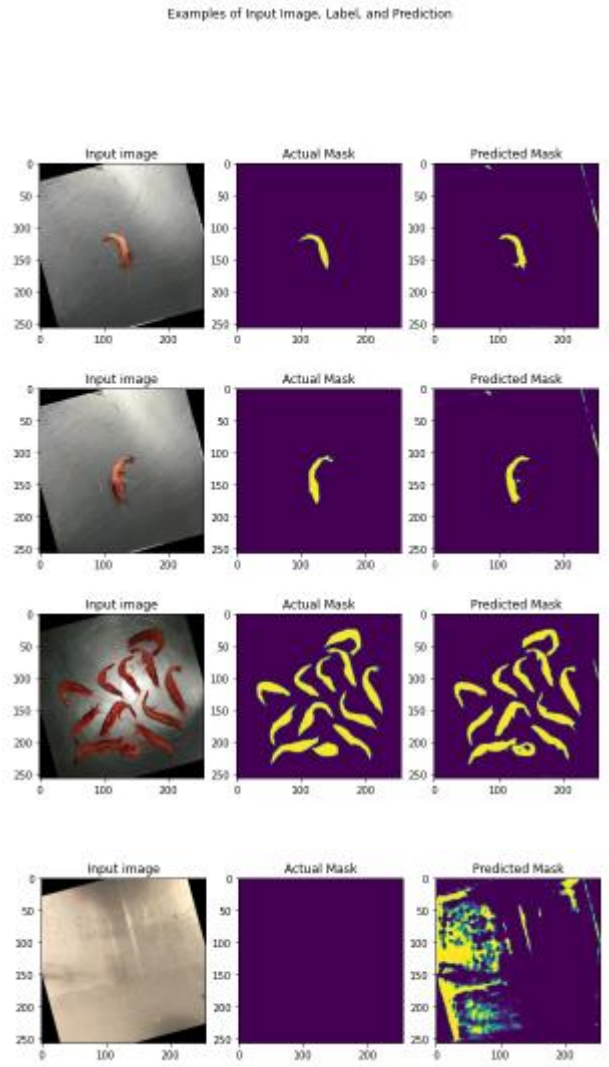


Figura 52. Exemple d'imatges, les seves màscares i la seva predicció.

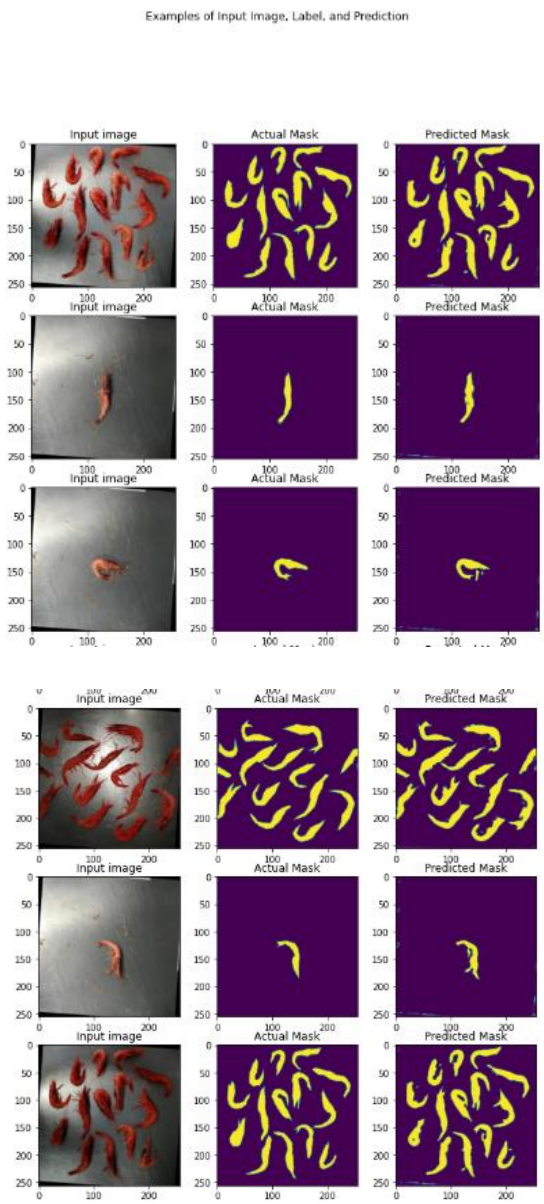


Figura 53. Exemple d'imatges, les seves màscares i la seva predicció.

El millor valor que pren la *validation dice loss* és de 0,10950, atorgant al model una puntuació de 89,05. Es pot observar que tot i augmentar significativament el número d'imatges, el resultat del model és pitjor. També es pot observar com en punts on no hauria de trobar cap gamba, ho fa. Per tal de solucionar aquest problema, en el següent experiment es variaran els paràmetres.

5.4. Experiment 4

Aquest experiment s'ha utilitzat una base de dades formada única i exclusivament per imatges obtingudes en un entorn real. Les dimensions de la base de dades utilitzada són les següents:

- **198 imatges**. Aquestes imatges s'han dividit en 158 imatges de entrenament i 40 de validació.

Per a les imatges de test s'han utilitzat imatges obtingudes en un entorn real, ja que el propòsit de la xarxa es reconèixer les gambes en aquell entorn.

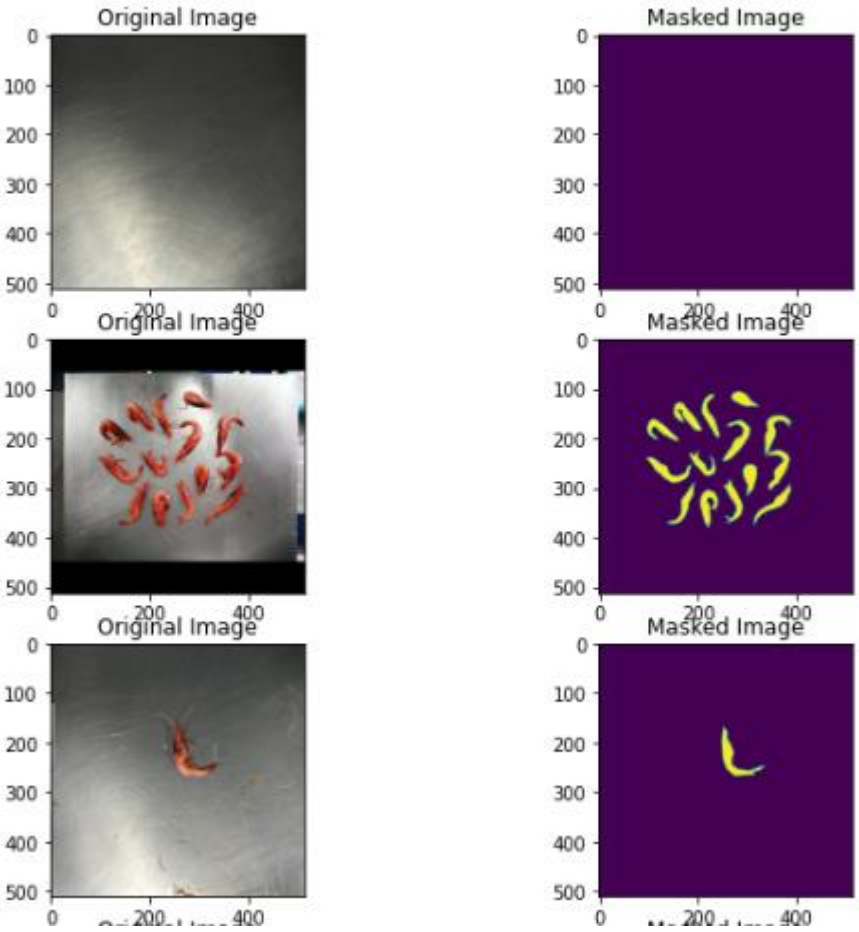


Figura 54. Exemple d'imatges i les seves màscares.

El valor dels paràmetres emprats ha estat el següent:

- Mida de les imatges = (512, 512, 3)
- Batch size = 4
- Epochs = 300

En les figures 55 i 56 respectivament, es mostren els resultats obtinguts en l'entrenament del model amb la base de dades i paràmetres anteriorment descrits, així como l'evolució de la funció de pèrdua.

```
Epoch 300/300
39/40 [============================>.] - ETA: 0s - loss: 0.1035 - dice_loss: 0.0861
Epoch 00300: val_dice_loss did not improve from 0.09164
40/40 [==============================] - 38s 955ms/step - loss: 0.1044 - dice_loss: 0.0868 - val_loss: 0.1119 - val_dice_loss: 0.0963
```

Figura 55. Últim epoch de l'entrenament

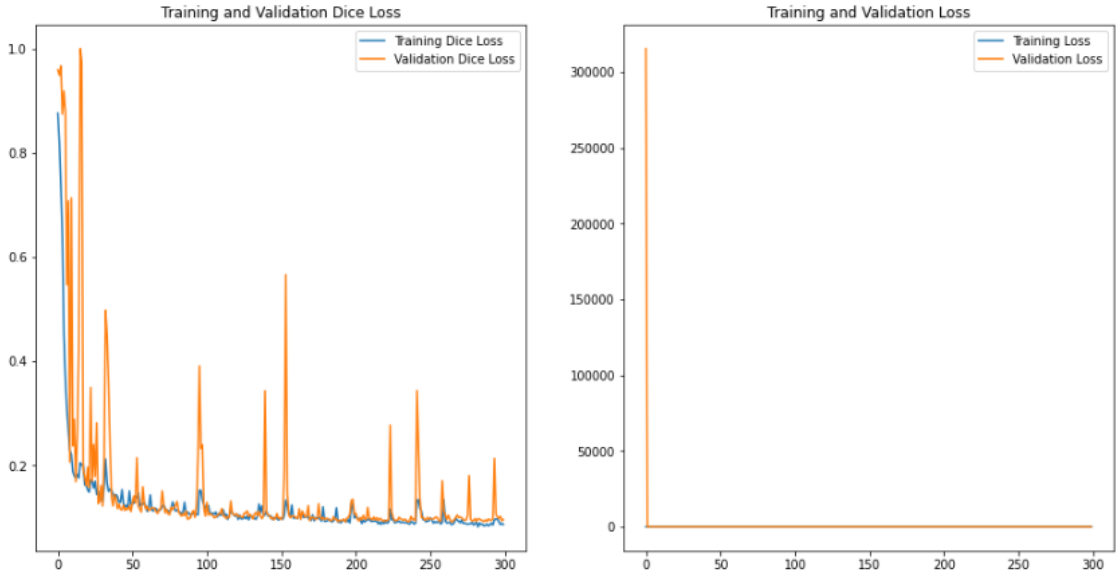


Figura 56. Evolució del valor de la funció de pèrdua.

En les figures 57 i 58 es mostren les prediccions de les màscares fetes per la xarxa de les imatges de validació i test respectivament.

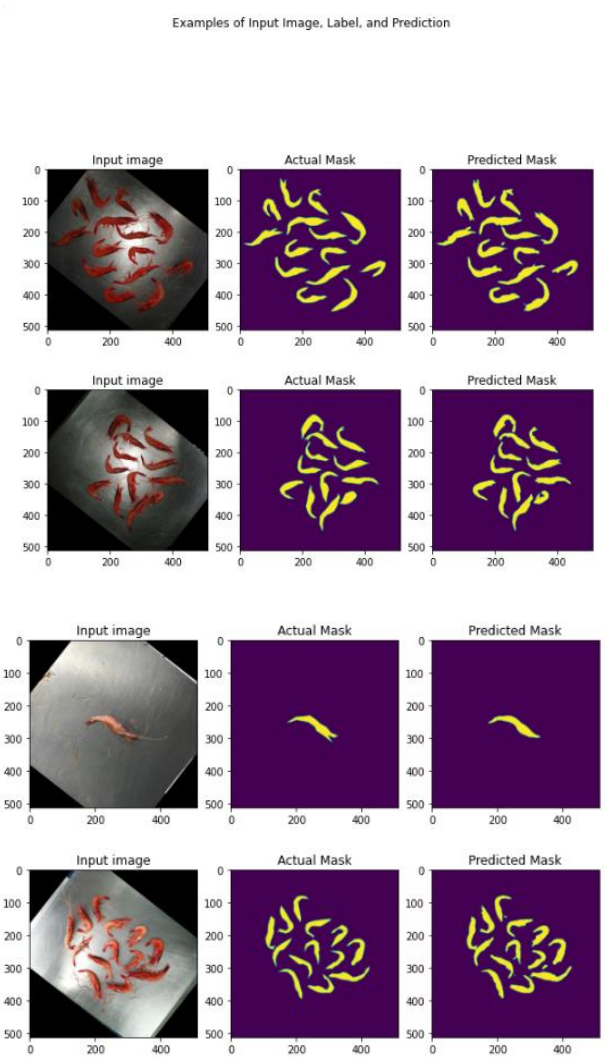


Figura 57. Exemple d'imatges, les seves màscares i la seva predicció.

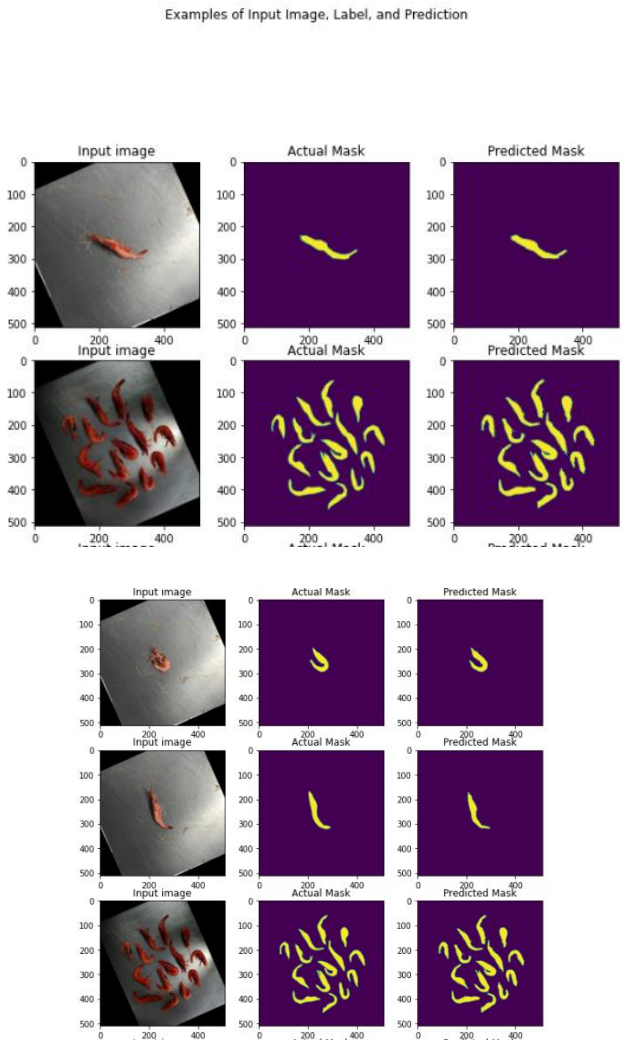


Figura 58. Exemple d'imatges, les seves màscares i la seva predicció.

El millor valor que pren la *validation dice loss* és de 0,09164, atorgant al model una puntuació de 90,83. Al augmentar la mida de la imatge i el número d'epochs, el model convergeix en un resultat millor. A més a més, com es pot observar, les prediccions ja no detecten gambes en espais buits com en l'experiment anterior.

5.5. Experiment 5

Aquest experiment s'ha utilitzat una base de dades formada única i exclusivament per imatges obtingudes en un entorn real. Les dimensions de la base de dades utilitzada són les següents:

- **263 imatges**. Aquestes imatges s'han dividit en 210 imatges de entrenament i 53 de validació.

Per a les imatges de test s'han utilitzat imatges obtingudes en un entorn real, ja que el propòsit de la xarxa es reconèixer les gambes en aquell entorn.

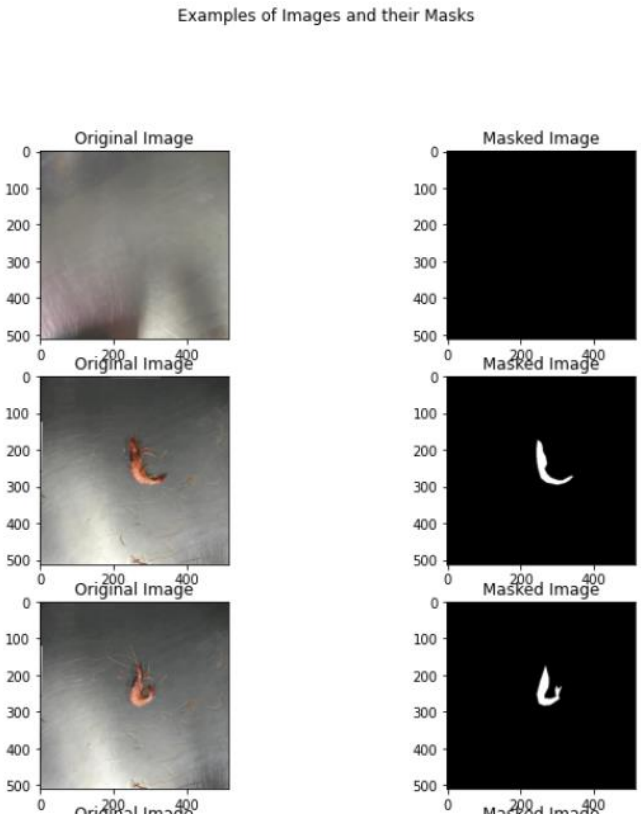


Figura 59. Exemple d'imatges i les seves màscares.

El valor dels paràmetres emprats ha estat el següent:

- Mida de les imatges = (512, 512, 3)
- Batch size = 4
- Epochs = 300

En les figures 60 i 61 respectivament, es mostren els resultats obtinguts en l'entrenament del model amb la base de dades i paràmetres anteriorment descrits, així como l'evolució de la funció de pèrdua.

```
Epoch 300/300
52/53 [============================>.] - ETA: 0s - loss: 0.1035 - dice_loss: 0.0785
Epoch 00300: val_dice_loss did not improve from 0.10544
53/53 [==============================] - 52s 984ms/step - loss: 0.1042 - dice_loss: 0.0789 - val_loss: 0.1861 - val_dice_loss: 0.1121
```

Figura 60. Últim epoch de l'entrenament

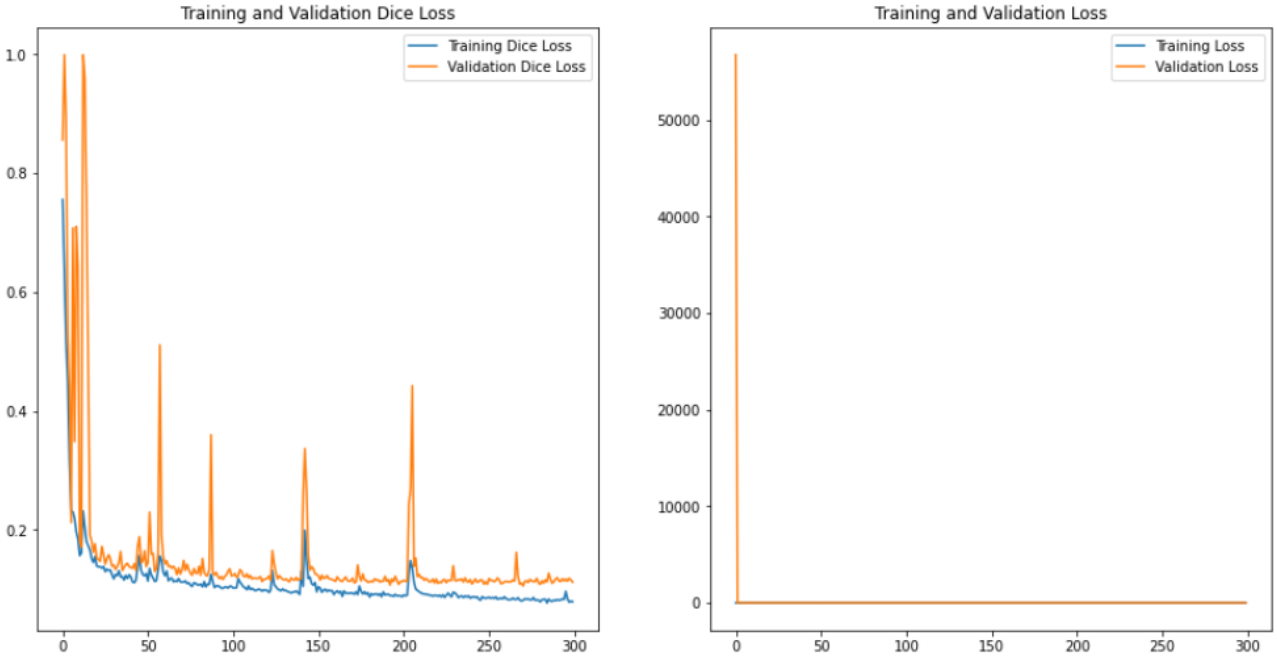


Figura 61. Evolució del valor de la funció de pèrdua.

En les figures 62 i 63 es mostren les prediccions de les màscares fetes per la xarxa de les imatges de validació i test respectivament.

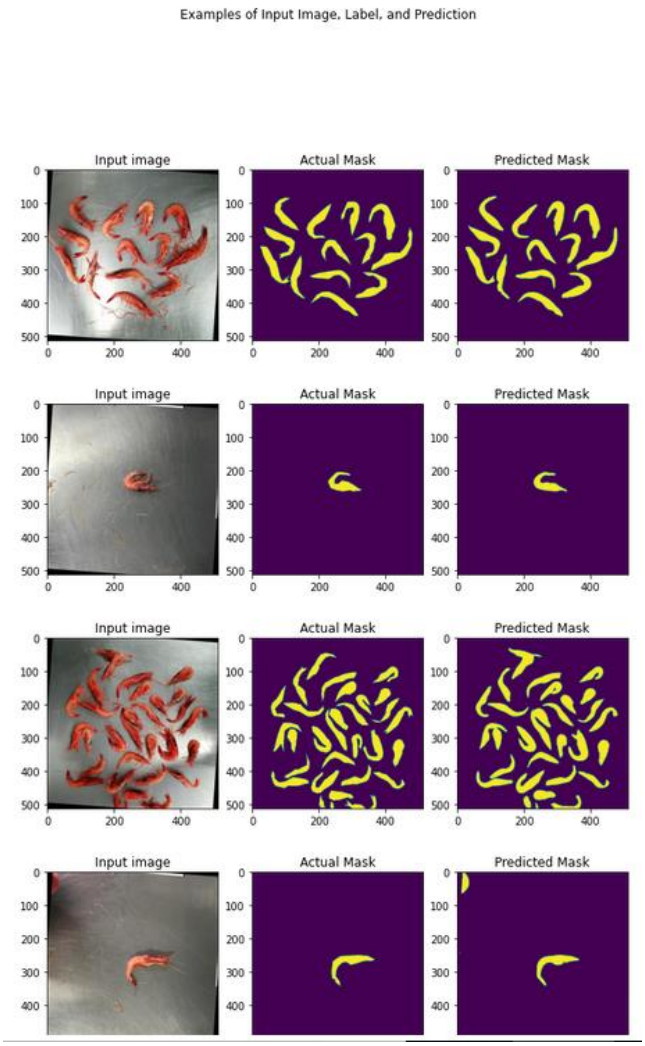


Figura 62. Exemple d'imatges, les seves màscares i la seva predicció.

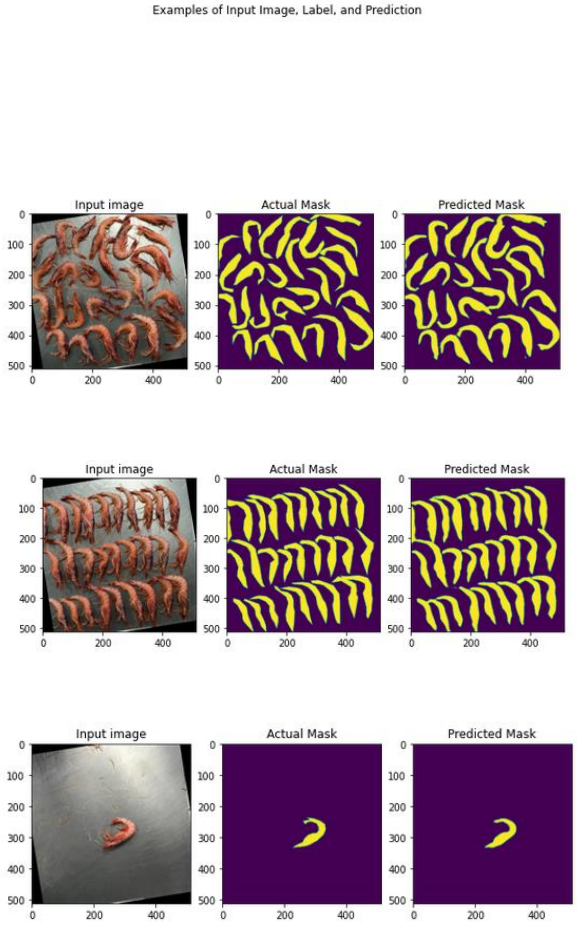


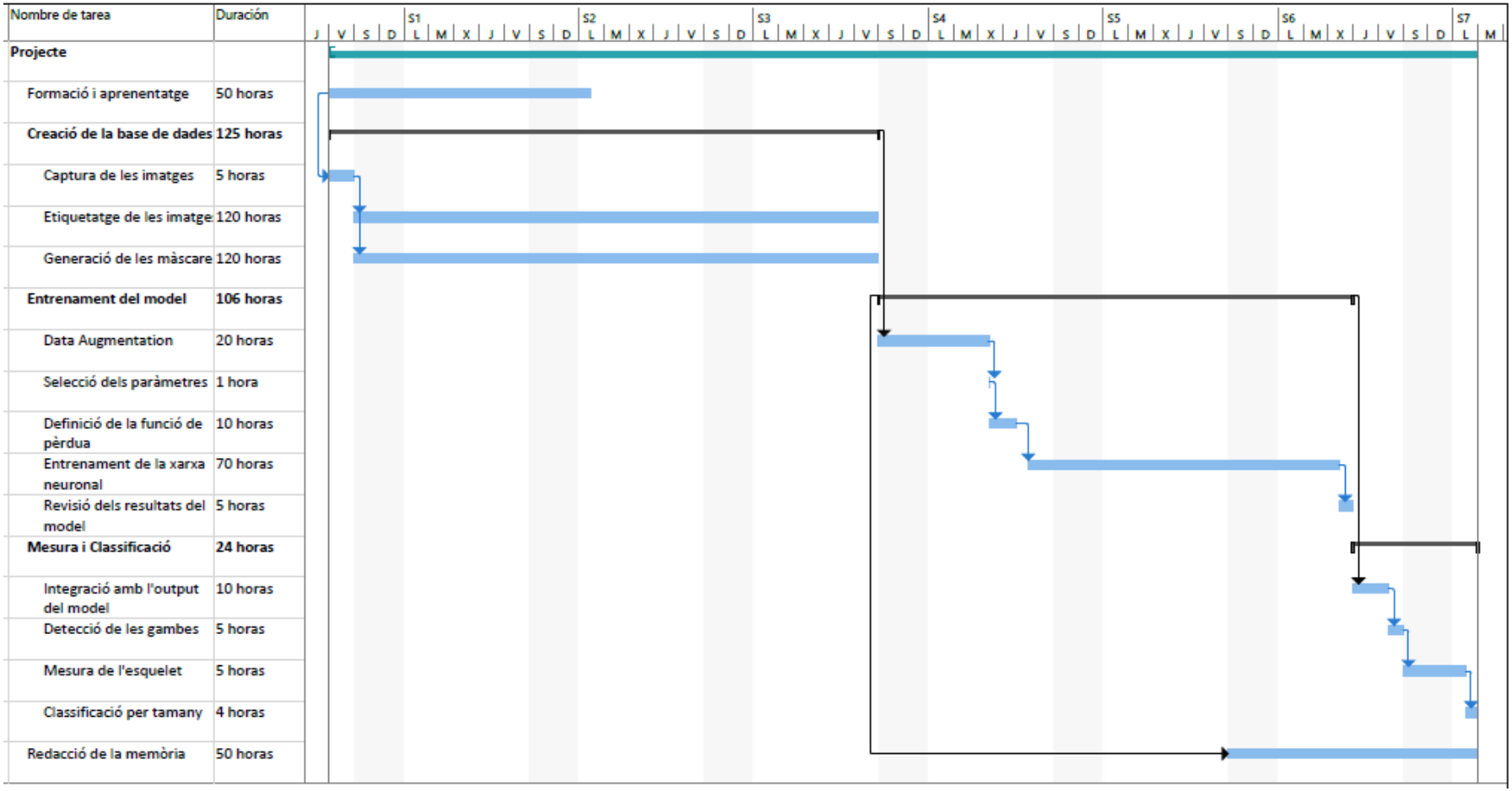
Figura 63. Exemple d'imatges, les seves màscares i la seva predicció.

El millor valor que pren la *validation dice loss* és de 0,10544, atorgant al model una puntuació de 89,45. Es pot observar que tot i augmentar el número d'imatges, en aquest experiment la puntuació del model ha estat inferior. Tot i això, observant les gràfiques de la figura 61 es pot veure com la variació entre la pèrdua en l'entrenament i en la validació és menor que en l'experiment 4.

La pitjor puntuació d'aquest experiment es pot explicar amb l'increment d'imatges que ha patit la base de dades. Les imatges afegides han estat imatges amb molts exemplars, on aquests es tocaven entre ells dificultant l'etiquetatge. Per tant, a l'hora de validar el model, es pot donar que aquest detecti gambes correctament, però a les màscares no estiguin senyalades com a tal.

6. Planificació

A continuació es mostra el diagrama de Gantt on es pot veure la planificació de les diferents tasques del projecte.



7. Anàlisi de costos

En aquest apartat es mostren detalladament els costos associats al projecte.

7.1. Costos de personal

El gran gruix dels costos derivats de la realització del projecte ve donat per la mà d'obra. Del total de 355 hores invertides la tasca que més temps ha ocupat ha sigut la creació de la base de dades.

Concepte / Tasca	Professional	Quantitat	Cost unitari	Cost
Formació i aprenentatge	Enginyer	50 h.	0 €/h	0 €
Creació de la base de dades	Enginyer	125 h.	35 €/h	4.375 €
Entrenament del model	Enginyer	106 h.	35 €/h	3.710 €
Mesura i Classificació	Enginyer	24 h.	35 €/h	840 €
Redacció de la memòria	Enginyer	50 h.	35 €/h	1.750 €
TOTAL		**355h.**		**10.675 €**

Taula 1. Costos de personal.

7.2. Costos de Software

Els costos de Software estan relacionats amb els costos d'ús de les llicències dels programes utilitzats durant aquest projecte.

Donat que totes les eines que s'han utilitzat durant la realització d'aquest projecte son software lliure i codi obert, el seu cost ha estat de 0 €.

7.3. Costos de Hardware

Els costos de Hardware són els relacionats amb la compra de l'equip necessari per a la realització d'aquest projecte. En aquest cas, s'ha utilitzat un portàtil Lenovo 300-15ISK.

Concepte	Professional	Quantitat	Cost unitari	Cost
Ordinador portàtil (Lenovo 300-15ISK)		1 u.	673 €/u	673 €
TOTAL		**1 u.**		**673 €**

Taula 2. Costos de hardware.

7.4. Costos Totals

A la taula 3 es presenta el resum dels costos associats a la realització d'aquest projecte.

Concepte	Cost
Costos de personal	10.675 €
Costos de Software	0 €
Costos de Hardware	673 €
TOTAL	**11.348 €**

Taula 3. Costos totals.

8. Anàlisi de l'impacte mediambiental

Al tractar-se d'un projecte emmarcat en un altre més ambiciós, no es generaran canvis susceptibles directes per al medi ambient, així com cap mena de residu contaminant. Tot i això, cal mencionar que aquest projecte servirà com a habilitador per a la implementació futura en un entorn real que si pugui tenir efectes directes en el medi ambient.

En el supòsit de la implementació de l'automatització del procés de reconeixement, classificació, recompte i mesura d'espècies, aquesta tindria un impacte positiu en el medi ambient, ja que es tractaria d'una eina que afavoriria a la transparència del sector i permetria la identificació de la pesca d'espècies protegides o d'exemplars que no superin les talles mínimes establertes per la legislació.

D'altra banda cal mencionar que el dispositiu emprat en aquest projecte (ordinador portàtil) consumeix electricitat, i està composat per elements que en el seu procés d'obtenció si que influeixen negativament en el medi ambient.

Tot i l'indicat en els paràgrafs anteriors, es considerarà negligible l'impacte ambiental d'aquest projecte.

Conclusions

Mitjançant aquest projecte es pretenia automatitzar el reconeixement de gambes vermelles i l'automatització del recompte i de la mesura dels individus reconeguts. Per tal de donar resposta a aquestes qüestions s'han hagut d'assolir altres objectius menors prèviament.

En primer lloc, es pot afirmar que la creació d'una base de dades s'ha assolit completament. Aquest era un requisit indispensable, ja que sense assolir-lo no hagués sigut possible dur a terme els següents passos del projecte. Cal destacar que tot i haver-se assolit, sempre es pot ampliar i millorar la base de dades construïda en aquest projecte.

En segon lloc, la implementació de l'algoritme de *data augmentation* també s'ha assolit completament. Gràcies a això, s'ha pogut ampliar el número d'imatges amb les que s'ha entrenat a la xarxa neuronal.

En tercer lloc, el primer dels objectius principals s'ha assolit completament: l'automatització del reconeixement de gambes vermelles. Tot i que el màxim de puntuació (90,83 en l'experiment) que s'ha obtingut en els diferents experiments realitzats, es considera que ha estat un èxit. Aquesta diferència es considera que és deguda a que el model identifica parts de les gambes que a l'hora de l'etiquetatge s'han negligit (potes i bigotis), o que s'han passat per alt degut a l'error humà.

Per últim, l'automatització del recompte i de la mesura dels individus reconeguts s'ha assolit parcialment. Si bé es cert que s'ha aconseguit fer un recompte precís dels elements identificats i posteriorment mesurar-los, s'han observat les següents problemàtiques:

- En el cas de trobar-se dues o mes gambes solapades o trobar-se fragments de gamba, el model ha sigut incapaç de distingir-ho d'un exemplar complet. Aquest fet és degut a que s'està treballant amb una segmentació semàntica i només es distingeix entre gamba o *background*, no s'identifiquen exemplars.
- El resultat de la mesura no s'expressa en unitats mètriques, ja que per tal de fer la conversió s'hauria d'haver introduït un element amb mida coneguda per tal de poder fer la conversió.

Els resultats obtinguts al llarg d'aquest treball han estat molt prometedors, és per això que s'anima a continuar i millorar quest projecte per tal de construir un pilot completament operacional.

Treballs Futurs

Donat els prometedors resultats d'aquest projecte i a que està emmarcat en un projecte amb un abast més ambiciós, existeixen una sèrie de treballs que no s'han dut a terme. Per tal de completar la implementació d'aquest projecte en un entorn real es proposen les següents tasques:

- **Inclusió d'un *threshold*.** Incloure en el model un *threshold* per tal de que la xarxa sigui capaç de detectar exemplars superposats o fragments que no puguin ser considerats un exemplar sencer.
- **Millora de l'algoritme de mesura**. Actualment l'algoritme de mesura classifica les gambes en tres categories segons la mida. L'objectiu és modificar i adaptar el codi, de tal manera que sigui capaç de mesurar els centímetres de cada exemplar.
- **Implementació en un dispositiu Jetson integrat amb una càmera fixe**. Dur terme la integració del model construït en aquest projecte en un dispositiu Jetson connectat a una càmera fixe i automatitzar el procés per tal d'estar en disposició d'un pilot completament operacional.

Un cop dutes a terme aquestes tasques, s'estarà en disposició d'implementar el pilot resultant en un entorn real per a tal de provar la seva eficàcia i impacte.

Bibliografia

[1] STACKOVERFLOW. Developer community.

[https://stackoverflow.com/, 19 de setembre de 2020]

[2] Python Package Index.

[https://pypi.org, 19 de setembre de 2020]

[3] Python Documentation.

[https://docs.python.org/3/, 20 de setembre de 2020]

[4] GitHub. The world's leading software development platform.

[https://github.com/, 19 de setembre de 2020]

[5] TensorFlow.

[https://www.tensorflow.org/, 19 de setembre de 2020]

[6] Keras. The Python deep learning API.

[https://keras.io/, 20 de setembre de 2020]

[7] Google Colab.

[https://colab.research.google.com, 20 de setembre de 2020]

[8] Project Jupyer.

[https://jupyter.org, 20 de setembre de 2020]

[9] LabelMe. The open annotation tool.

[https://labelme.csail.mit.edu/Release3.0/, 20 de setembre de 2020]

[10] Medium. Towards data science

[https://towardsdatascience.com/understanding-semantic-segmentation-with-unet-6be4f42d4b47, 19 de setembre de 2020]

[11] Kagel. TGS Salt Identification Challenge.

[https://www.kaggle.com/c/tgs-salt-identification-challenge, 20 de setembre de 2020]

[12] Kagel. Carvana Image Masking Challenge.

[https://www.kaggle.com/c/carvana-image-masking-challenge, 20 de setembre de 2020]

[13] Fausto Milletari, Nassir Navab, Seyed-Ahmad Ahmadi (2016). *V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation.*

[http://campar.in.tum.de/pub/milletari2016Vnet/milletari2016Vnet.pdf, 19 de setembre de 2020]

[14] Platzi. Save the data.

[https://www.youtube.com/channel/UC55-mxUj5Nj3niXFReG44OQ, 20 de setembre de 2020]

[15] 3Blue1Brown. Neural networks.

[https://www.youtube.com/playlist?list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi, 20 de setembre de 2020]

[16] APM Tech.

[https://www.youtube.com/watch?v=ns2L2T6wvAY, 20 de setembre de 2020]

[17] A collaborative report by DHL and IBM on implications and use case for the logistics industry (2018). *Artificial Intelligence in Logistics*.

[https://www.dhl.com/content/dam/dhl/global/core/documents/pdf/glo-core-trend-report-artificial-intelligence.pdf, 20 de setembre de 2020]

Annex de Codi

U-Net

In [ ]:
```
species = './Gambes'

!mkdir Gambes
!cd Gambes

from google.colab import drive
drive.mount('/content/drive')

!ls -l ./drive/My\ Drive/Gambes

!cp ./drive/My\ Drive/Gambes/train.zip ./Gambes/
!cp ./drive/My\ Drive/Gambes/train_masks.zip ./Gambes/
!cp ./drive/My\ Drive/Gambes/test.zip ./Gambes/
!cp ./drive/My\ Drive/Gambes/test_masks.zip ./Gambes/
```

In [ ]:
```
img_shape = (512, 512, 3)
batch_size = 4
epochs = 300
```

In [ ]:
```
%tensorflow_version 1.x
```

In [ ]:
```
import os
import glob
import zipfile
import functools
from os.path import join, isfile
import random
import math

import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
mpl.rcParams['axes.grid'] = False
mpl.rcParams['figure.figsize'] = (12,12)

from sklearn.model_selection import train_test_split
import matplotlib.image as mpimg
import pandas as pd
from PIL import Image

import tensorflow as tf
import tensorflow.contrib as tfcontrib
from tensorflow.python.keras import layers
from tensorflow.python.keras import losses
from tensorflow.python.keras import models
from tensorflow.python.keras import backend as K

import shelve
import dill
```

In [ ]:
```
def load_data_from_zip(folder, file):
  with zipfile.ZipFile(os.path.join(species, file), "r") as zip_ref:
    unzipped_file = zip_ref.namelist()[0]
    zip_ref.extractall(species)
```

In [ ]:
```
load_data_from_zip(species, 'train.zip')
```

In [ ]:
```
load_data_from_zip(species, 'test.zip')
```

In [ ]:
```
load_data_from_zip(species, 'train_masks.zip')
```

In [ ]:
```
load_data_from_zip(species, 'test_masks.zip')
```

In [ ]:
```
train_img_dir = os.path.join(species, 'train')
train_mask_dir = os.path.join(species, 'train_masks')
test_dir = os.path.join(species, 'test')
test_mask_dir = os.path.join(species, 'test_masks')
```

In [ ]:
```
x_train_filenames = [join(train_img_dir,f) for f in os.listdir(train_img_dir) if isfile(join(train_img_
dir, f))]
x_train_filenames.sort()

y_train_filenames = [join(train_mask_dir,f) for f in os.listdir(train_mask_dir) if isfile(join(train_ma
sk_dir, f))]
y_train_filenames.sort()


x_test_filenames = [join(test_dir,f) for f in os.listdir(test_dir) if isfile(join(test_dir, f))]
x_test_filenames.sort()

y_test_filenames = [join(test_mask_dir,f) for f in os.listdir(test_mask_dir) if isfile(join(test_mask_d
ir, f))]
y_test_filenames.sort()
```

In [ ]:
```
x_train_filenames, x_val_filenames, y_train_filenames, y_val_filenames = \
                    train_test_split(x_train_filenames, y_train_filenames, test_size=0.2, random_state=
42)
```

In [ ]:
```
num_train_examples = len(x_train_filenames)
num_val_examples = len(x_val_filenames)

print("Number of training examples: {}".format(num_train_examples))
print("Number of validation examples: {}".format(num_val_examples))
```

In [ ]:
```
display_num = 5

r_choices = np.random.choice(num_train_examples, display_num)

plt.figure(figsize=(10, 15))
for i in range(0, display_num * 2, 2):
  img_num = r_choices[i // 2]
  x_pathname = x_train_filenames[img_num]
  y_pathname = y_train_filenames[img_num]

  plt.subplot(display_num, 2, i + 1)
  plt.imshow(mpimg.imread(x_pathname))
  plt.title("Original Image")

  example_labels = Image.open(y_pathname)
  label_vals = np.unique(example_labels)

  plt.subplot(display_num, 2, i + 2)
  plt.imshow(example_labels)
  plt.title("Masked Image")

plt.suptitle("Examples of Images and their Masks")
plt.show()
```

In [ ]:
```
def _process_pathnames(fname, label_path):

  img_str = tf.read_file(fname)
  img = tf.image.decode_jpeg(img_str, channels=3)

  label_img_str = tf.read_file(label_path)

  label_img = tf.image.decode_gif(label_img_str)[0]

  label_img = label_img[:, :, 0]
  label_img = tf.expand_dims(label_img, axis=-1)
  return img, label_img
```

In [ ]:
```
def shift_img(output_img, label_img, width_shift_range, height_shift_range):
  """This fn will perform the horizontal or vertical shift"""
  if width_shift_range or height_shift_range:
      if width_shift_range:
        width_shift_range = tf.random_uniform([],
                                              -width_shift_range * img_shape[1],
                                              width_shift_range * img_shape[1])
      if height_shift_range:
        height_shift_range = tf.random_uniform([],
                                               -height_shift_range * img_shape[0],
                                               height_shift_range * img_shape[0])

      output_img = tfcontrib.image.translate(output_img,
                                             [width_shift_range, height_shift_range])
      label_img = tfcontrib.image.translate(label_img,
                                            [width_shift_range, height_shift_range])
  return output_img, label_img
```

In [ ]:
```
def flip_img(horizontal_flip, tr_img, label_img):
  if horizontal_flip:
    flip_prob = tf.random_uniform([], 0.0, 1.0)
    tr_img, label_img = tf.cond(tf.less(flip_prob, 0.5),
                                lambda: (tf.image.flip_left_right(tr_img), tf.image.flip_left_right(lab
el_img)),
                                lambda: (tr_img, label_img))
    flip_prob = tf.random_uniform([], 0.0, 1.0)
    tr_img, label_img = tf.cond(tf.less(flip_prob, 0.5),
                                lambda: (tf.image.flip_up_down(tr_img), tf.image.flip_up_down(label_img
)),
                                lambda: (tr_img, label_img))

  return tr_img, label_img
```

In [ ]:
```
def rotate_img(tr_img, label_img):
  degrees = random.randint(1,359)

  tr_img = tf.contrib.image.rotate(tr_img, degrees * math.pi / 180, interpolation='BILINEAR')
  label_img = tf.contrib.image.rotate(label_img, degrees * math.pi / 180, interpolation='BILINEAR')



  return tr_img, label_img
```

In [ ]:
```
def _augment(img,
             label_img,
             resize=None,
             pad_crop=False,
             scale=1,
             hue_delta=0,
             horizontal_flip=False,
             width_shift_range=0,
             height_shift_range=0):

  #if pad_crop is True:
    # Resize both images
  #  label_img = tf.image.resize_images(label_img, [90,90])
  #  label_img = tf.image.resize_image_with_crop_or_pad(label_img, target_height=256, target_width=256)
  #  img = tf.image.resize_images(img, [90,90])
  #  img = tf.image.resize_image_with_crop_or_pad(img, target_height=256, target_width=256)

  #if pad_crop is False:
   #  Resize both images
    #label_img = tf.image.resize_images(label_img, [256,256])
    #img = tf.image.resize_images(img, [256,256])

  if hue_delta:
    img = tf.image.random_hue(img, hue_delta)

  img, label_img = flip_img(horizontal_flip, img, label_img)
  img, label_img = shift_img(img, label_img, width_shift_range, height_shift_range)
  img, label_img = rotate_img(img, label_img)
  label_img = tf.to_float(label_img) * scale
  img = tf.to_float(img) * scale
  return img, label_img
```

In [ ]:
```
def get_baseline_dataset(filenames,
                         labels,
                         preproc_fn=functools.partial(_augment),
                         threads=5,
                         batch_size=batch_size,
                         shuffle=True):
  num_x = len(filenames)

  dataset = tf.data.Dataset.from_tensor_slices((filenames, labels))

  dataset = dataset.map(_process_pathnames, num_parallel_calls=threads)
  if preproc_fn.keywords is not None and 'resize' not in preproc_fn.keywords:
    assert batch_size == 1, "Batching images must be of the same size"

  dataset = dataset.map(preproc_fn, num_parallel_calls=threads)

  if shuffle:
    dataset = dataset.shuffle(num_x)



  dataset = dataset.repeat().batch(batch_size)
  return dataset
```

In [ ]:
```
tr_cfg = {
    'resize': [img_shape[0], img_shape[1]],
    'pad_crop': True,
    'scale': 1 / 255.,
    'hue_delta': 0.1,
    'horizontal_flip': True,
    'width_shift_range': 0.1,
    'height_shift_range': 0.1
}
tr_preprocessing_fn = functools.partial(_augment, **tr_cfg)
```

In [ ]:
```
val_cfg = {
    'resize': [img_shape[0], img_shape[1]],
    'pad_crop': True,
    'scale': 1 / 255.,
}
val_preprocessing_fn = functools.partial(_augment, **val_cfg)
```

In [ ]:
```
test_cfg = {
    'resize': [img_shape[0], img_shape[1]],
    'pad_crop': False,
    'scale': 1 / 255.,
}
test_preprocessing_fn = functools.partial(_augment, **test_cfg)
```

In [ ]:
```
train_ds = get_baseline_dataset(x_train_filenames,
                                y_train_filenames,
                                preproc_fn=tr_preprocessing_fn,
                                batch_size=batch_size)
```

In [ ]:
```
val_ds = get_baseline_dataset(x_val_filenames,
                              y_val_filenames,
                              preproc_fn=val_preprocessing_fn,
                              batch_size=batch_size)
```

In [ ]:
```
test_ds = get_baseline_dataset(x_test_filenames,
                               y_test_filenames,
                               preproc_fn=test_preprocessing_fn,
                               batch_size=batch_size)
```

In [ ]:
```
temp_ds = get_baseline_dataset(x_train_filenames,
                               y_train_filenames,
                               preproc_fn=test_preprocessing_fn,
                               batch_size=1,
                               shuffle=False)

data_aug_iter = temp_ds.make_one_shot_iterator()
next_element = data_aug_iter.get_next()
with tf.Session() as sess:
  batch_of_imgs, label = sess.run(next_element)

  plt.figure(figsize=(10, 10))
  img = batch_of_imgs[0]

  plt.subplot(1, 2, 1)
  plt.imshow(img)

  plt.subplot(1, 2, 2)
  plt.imshow(label[0, :, :, 0])
  plt.show()
```

In [ ]:
```
def conv_block(input_tensor, num_filters):
  encoder = layers.Conv2D(num_filters, (3, 3), padding='same', kernel_initializer = 'he_normal')(input_
tensor)
  encoder = layers.BatchNormalization()(encoder)
  encoder = layers.Activation('relu')(encoder)
  encoder = layers.Conv2D(num_filters, (3, 3), padding='same', kernel_initializer = 'he_normal')(encode
r)
  encoder = layers.BatchNormalization()(encoder)
  encoder = layers.Activation('relu')(encoder)
  return encoder

def encoder_block(input_tensor, num_filters):
  encoder = conv_block(input_tensor, num_filters)
  encoder_pool = layers.MaxPooling2D((2, 2), strides=(2, 2))(encoder)

  return encoder_pool, encoder

def decoder_block(input_tensor, concat_tensor, num_filters):

  decoder = layers.UpSampling2D(size=(2,2),interpolation='bilinear')(input_tensor)

  decoder = layers.Conv2D(num_filters, (2, 2),activation = 'relu', padding='same', kernel_initializer =
 'he_normal')(decoder)

  decoder = layers.concatenate([concat_tensor, decoder], axis=3)

  decoder = layers.BatchNormalization()(decoder)

  decoder = layers.Conv2D(num_filters, (3, 3),activation = 'relu', padding='same', kernel_initializer =
 'he_normal')(decoder)
  decoder = layers.BatchNormalization()(decoder)

  decoder = layers.Conv2D(num_filters, (3, 3),activation = 'relu', padding='same', kernel_initializer =
 'he_normal')(decoder)
  decoder = layers.BatchNormalization()(decoder)

  return decoder
```

In [ ]:
```
inputs = layers.Input(shape=img_shape)
# 256

encoder1_pool, encoder1 = encoder_block(inputs, 64)
# 128

encoder2_pool, encoder2 = encoder_block(encoder1_pool, 128)
# 64

encoder3_pool, encoder3 = encoder_block(encoder2_pool, 256)
# 32

encoder4_pool, encoder4 = encoder_block(encoder3_pool, 512)
# 16

encoder5 = conv_block(encoder4_pool, 1024)
encoder5 = conv_block(encoder5, 1024)
encoder5 = layers.Dropout(0.5)(encoder5)
#16

decoder4 = decoder_block(encoder5, encoder4, 512)
# 32

decoder3 = decoder_block(decoder4, encoder3, 256)
# 64

decoder2 = decoder_block(decoder3, encoder2, 128)
# 128

decoder1 = decoder_block(decoder2, encoder1, 64)
# 256

outputs = layers.Conv2D(1, (1, 1), activation='sigmoid')(decoder1)
```

In [ ]:
```
model = models.Model(inputs=[inputs], outputs=[outputs])
```

In [ ]:
```
def dice_coeff(y_true, y_pred):
    smooth = 1.

    y_true_f = tf.reshape(y_true, [-1])
    y_pred_f = tf.reshape(y_pred, [-1])
    intersection = tf.reduce_sum(y_true_f * y_pred_f)
    score = (2. * intersection + smooth) / (tf.reduce_sum(y_true_f) + tf.reduce_sum(y_pred_f) + smooth)
    return score
```

In [ ]:
```
def dice_loss(y_true, y_pred):
    loss = 1 - dice_coeff(y_true, y_pred)
    return loss
```

In [ ]:
```
def bce_dice_loss(y_true, y_pred):
    loss = losses.binary_crossentropy(y_true, y_pred) + dice_loss(y_true, y_pred)
    return loss
```

In [ ]:
```
model.compile(optimizer='adam', loss=bce_dice_loss, metrics=[dice_loss])

model.summary()
```

In [ ]:
```
save_model_path = '/tmp/weights_def.hdf5'
```

In [ ]:
```
cp = tf.keras.callbacks.ModelCheckpoint(filepath=save_model_path, monitor='val_dice_loss', save_best_on
ly=True, verbose=1)
```

In [ ]:
```
history = model.fit(train_ds,
                    steps_per_epoch=int(np.ceil(num_train_examples / float(batch_size))),
                    epochs=epochs,
                    validation_data=val_ds,
                    validation_steps=int(np.ceil(num_val_examples / float(batch_size))),
                    callbacks=[cp])

!cp /tmp/weights_def.hdf5 ./drive/My\ Drive/Gambes/
!mkdir -p saved_model
model.save('saved_model/my_model')
```

In [ ]:
```
!cp /tmp/weights_def.hdf5 ./drive/My\ Drive/Gambes/
```

In [ ]:
```
dice = history.history['dice_loss']
val_dice = history.history['val_dice_loss']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(epochs)

plt.figure(figsize=(16, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, dice, label='Training Dice Loss')
plt.plot(epochs_range, val_dice, label='Validation Dice Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Dice Loss')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')

plt.show()
```

In [ ]:
```
!cp ./drive/My\ Drive/Gambes/weights_def.hdf5 /tmp/weights_def.hdf5
```

In [ ]:
```
model = models.load_model(save_model_path, custom_objects={'bce_dice_loss': bce_dice_loss,
                                                           'dice_loss': dice_loss})
```

In [ ]:
```
from keras.preprocessing.image import save_img
```

In [ ]:
```
data_aug_iter = val_ds.make_one_shot_iterator()
next_element = data_aug_iter.get_next()


plt.figure(figsize=(10, 20))
for i in range(5):
  batch_of_imgs, label = tf.keras.backend.get_session().run(next_element)
  img = batch_of_imgs[0]
  predicted_label = model.predict(batch_of_imgs)[0]



  save_img('/tmp/img.png',img)
  save_img('/tmp/mask.png',predicted_label)




  plt.subplot(5, 3, 3 * i + 1)
  plt.imshow(img)
  plt.title("Input image")

  plt.subplot(5, 3, 3 * i + 2)
  plt.imshow(label[0, :, :, 0])
  plt.title("Actual Mask")
  plt.subplot(5, 3, 3 * i + 3)
  plt.imshow(predicted_label[:, :, 0])
  plt.title("Predicted Mask")
plt.suptitle("Examples of Input Image, Label, and Prediction")
plt.show()
```

In [ ]:
```
test_aug_iter = test_ds.make_one_shot_iterator()
next_element = test_aug_iter.get_next()


plt.figure(figsize=(10, 20))
n=6

for i in range(n):
  batch_of_imgs, label = tf.keras.backend.get_session().run(next_element)
  img = batch_of_imgs[0]
  predicted_label = model.predict(batch_of_imgs)[0]

  plt.subplot(n, 3, 3 * i + 1)
  plt.imshow(img)
  plt.title("Input image")

  plt.subplot(n, 3, 3 * i + 2)
  plt.imshow(label[0, :, :, 0])
  plt.title("Actual Mask")
  plt.subplot(n, 3, 3 * i + 3)
  plt.imshow(predicted_label[:, :, 0])
  plt.title("Predicted Mask")


plt.suptitle("Examples of Input Image, Label, and Prediction")
plt.show()
```

In [ ]:
```
test_aug_iter = test_ds.make_one_shot_iterator()
next_element = test_aug_iter.get_next()


plt.figure(figsize=(10, 20))

for i in range(46):
  batch_of_imgs, label = tf.keras.backend.get_session().run(next_element)
  img = batch_of_imgs[0]
  predicted_label = model.predict(batch_of_imgs)[0]

  l = (label*255).astype(np.uint8)
  label_PIL = Image.fromarray(l[0, :, :, 0],mode='L')
  label_PIL.save('label_'+str(i)+'.png')

  pl = (predicted_label * 255).astype(np.uint8)
  pl_PIL = Image.fromarray(pl[:, :, 0], mode = 'L')
  pl_PIL.save('predicted_label_'+str(i)+'.png')
```

Identificació i mesura

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
%tensorflow_version 1.x
```

In [ ]:
```
import os
import glob
import zipfile
import functools
from os.path import join, isfile

import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
mpl.rcParams['axes.grid'] = False
mpl.rcParams['figure.figsize'] = (12,12)

from sklearn.model_selection import train_test_split
import matplotlib.image as mpimg
import pandas as pd
from PIL import Image

import tensorflow as tf
import tensorflow.contrib as tfcontrib
from tensorflow.python.keras import layers
from tensorflow.python.keras import losses
from tensorflow.python.keras import models
from tensorflow.python.keras import backend as K

import shelve
import dill
```

In [ ]:
```
def bce_dice_loss(y_true, y_pred):
    loss = losses.binary_crossentropy(y_true, y_pred) + dice_loss(y_true, y_pred)
    return loss

def dice_loss(y_true, y_pred):
    loss = 1 - dice_coeff(y_true, y_pred)
    return loss

def dice_coeff(y_true, y_pred):
    smooth = 1.
    # Flatten
    y_true_f = tf.reshape(y_true, [-1])
    y_pred_f = tf.reshape(y_pred, [-1])
    intersection = tf.reduce_sum(y_true_f * y_pred_f)
    score = (2. * intersection + smooth) / (tf.reduce_sum(y_true_f) + tf.reduce_sum(y_pred_f) + smooth)
    return score
```

In [ ]:
```
from keras.models import load_model
from keras.preprocessing import image
import matplotlib.pyplot as plt
import numpy as np
import os


def load_image(img_path, show=False):

    img = image.load_img(img_path, target_size=(256, 256))
    img_tensor = image.img_to_array(img)
    img_tensor = np.expand_dims(img_tensor, axis=0)
    img_tensor /= 255.

    if show:
        plt.imshow(img_tensor[0])
        plt.axis('off')
        plt.show()

    return img_tensor

if __name__ == "__main__":


    save_model_path = '/tmp/weights_def.hdf5'
    !cp ./drive/My\ Drive/Gambes/weights_def.hdf5 /tmp/weights_def.hdf5
    model = models.load_model(save_model_path, custom_objects={'bce_dice_loss': bce_dice_loss,
                                                               'dice_loss': dice_loss})


    img_path = '/content/drive/My Drive/Gambes/img_1.png'


    new_image = load_image(img_path)


    pred = model.predict(new_image)
```

In [ ]:
```
plt.axis('off')
plt.imshow(pred[0][:, :, 0])
plt.savefig('/content/drive/My Drive/Gambes/img_result.png',bbox_inches='tight', pad_inches=0)
```

In [ ]:
```
image_file = Image.open("/content/drive/My Drive/Gambes/img_result.png")
image_file = image_file.convert(mode='1',dither=Image.NONE)
image_file.save('/content/drive/My Drive/Gambes/img_1.png')
```

In [ ]:
```
!pip install skan
```

In [ ]:
```
from skimage import io, filters, measure, color
from scipy import ndimage
from skimage.morphology import skeletonize
from skimage import data
import matplotlib.pyplot as plt
from skimage.util import invert
from skan import Skeleton, summarize
```

In [ ]:
```
img = io.imread("/content/drive/My Drive/Gambes/img_1.png")
dic_label = {}
dic_dlabel = {}

labeled_mask, num_labels = ndimage.label(img, structure=[[1,1,1],[1,1,1],[1,1,1]])
clusters = measure.regionprops(labeled_mask, img)
j = 0
dic_img={}
dic_dist={}
while j < len(clusters):
    img_sk = clusters[j].image
    img_sk = img_sk.astype(int)
    skeleton = skeletonize(img_sk)
    try:
        branch_data = summarize(Skeleton(skeleton))
        dic_img['gamba_'+str(j)] = {'dist':max(branch_data['branch-distance']), 'data':branch_data, 'cl
uster':clusters[j]}
        dic_dist['gamba_'+str(j)] = max(branch_data['branch-distance'])
    except ValueError:
        pass
    j = j+1
dic_label['label_'] = {'img':dic_img, 'dist':dic_dist}
dic_dlabel['label_']=dic_dist
```

In [ ]:
```
dic_dlabel = dic_dlabel['label_']
sum = 0
for value in dic_dlabel.values():
  sum = sum + float(value)
mitja = sum / len(dic_dlabel.values())
```

In [ ]:
```
Grans = 0
Petits = 0
Mitjans = 0
for value in dic_dlabel.values():
  if float(value) > mitja * 1.2:
    Grans = Grans + 1
  elif float(value) < mitja * 0.8:
    Petits = Petits + 1
  else:
    Mitjans = Mitjans + 1

d_classificació = {'Grans': Grans, 'Mitjans': Mitjans, 'Petits': Petits}
d_classificació
```

Inferència

In [6]:
```
model_path = '/tmp/weights_def_e3.hdf5'
#plt.imshow(pred[0][:, :, 0])
img = '/content/drive/My Drive/Gambes/img_1.png'
def inferencia(img_path, model_path):
  import warnings
  warnings.filterwarnings("ignore")
  %tensorflow_version 1.x
  import os
  import glob
  import zipfile
  import functools
  from os.path import join, isfile
  import numpy as np
  import matplotlib.pyplot as plt
  import matplotlib as mpl
  mpl.rcParams['axes.grid'] = False
  mpl.rcParams['figure.figsize'] = (12,12)
  from sklearn.model_selection import train_test_split
  import matplotlib.image as mpimg
  import pandas as pd
  from PIL import Image
  import tensorflow as tf
  import tensorflow.contrib as tfcontrib
  from tensorflow.python.keras import layers
  from tensorflow.python.keras import losses
  from tensorflow.python.keras import models
  from tensorflow.python.keras import backend as K
  import shelve
  import dill
  !pip install skan
  from skimage import io, filters, measure, color
  from scipy import ndimage
  from skimage.morphology import skeletonize
  from skimage import data
  import matplotlib.pyplot as plt
  from skimage.util import invert
  from skan import Skeleton, summarize


  def bce_dice_loss(y_true, y_pred):
    loss = losses.binary_crossentropy(y_true, y_pred) + dice_loss(y_true, y_pred)
    return loss

  def dice_loss(y_true, y_pred):
      loss = 1 - dice_coeff(y_true, y_pred)
      return loss
```

```
  def dice_coeff(y_true, y_pred):
      smooth = 1.
      y_true_f = tf.reshape(y_true, [-1])
      y_pred_f = tf.reshape(y_pred, [-1])
      intersection = tf.reduce_sum(y_true_f * y_pred_f)
      score = (2. * intersection + smooth) / (tf.reduce_sum(y_true_f) + tf.reduce_sum(y_pred_f) + smoot
h)
      return score
  from keras.models import load_model
  from keras.preprocessing import image
  import matplotlib.pyplot as plt
  import numpy as np
  import os
  def load_image(img_path, show=False):
      img = image.load_img(img_path, target_size=(512, 512))
      img_tensor = image.img_to_array(img)
      img_tensor = np.expand_dims(img_tensor, axis=0)
      img_tensor /= 255.

      if show:
          plt.imshow(img_tensor[0])
          plt.axis('off')
          plt.show()

      return img_tensor

  if __name__ == "__main__":
      save_model_path = model_path
      !cp ./drive/My\ Drive/Gambes/weights_def_e3.hdf5 /tmp/weights_def_e3.hdf5
      model = models.load_model(save_model_path, custom_objects={'bce_dice_loss': bce_dice_loss,
                                                                 'dice_loss': dice_loss})
      new_image = load_image(img_path)
      pred = model.predict(new_image)
```

```
  plt.axis('off')
  plt.imshow(pred[0][:, :, 0])
  plt.savefig('/content/drive/My Drive/Gambes/img_result.png',bbox_inches='tight', pad_inches=0)
  image_file = Image.open("/content/drive/My Drive/Gambes/img_result.png")
  image_file = image_file.convert(mode='1',dither=Image.NONE)
  image_file.save(img_path)
  img = io.imread(img_path)
  dic_label = {}
  dic_dlabel = {}
  labeled_mask, num_labels = ndimage.label(img, structure=[[1,1,1],[1,1,1],[1,1,1]])
  clusters = measure.regionprops(labeled_mask, img)
  j = 0
  dic_img={}
  dic_dist={}
  while j < len(clusters):
      img_sk = clusters[j].image
      img_sk = img_sk.astype(int)
      skeleton = skeletonize(img_sk)
      try:
          branch_data = summarize(Skeleton(skeleton))
          dic_img['gamba_'+str(j)] = {'dist':max(branch_data['branch-distance']), 'data':branch_data,
'cluster':clusters[j]}
          dic_dist['gamba_'+str(j)] = max(branch_data['branch-distance'])
      except ValueError:
          pass
      j = j+1
  dic_label['label_'] = {'img':dic_img, 'dist':dic_dist}
  dic_dlabel['label_']=dic_dist
  dic_dlabel = dic_dlabel['label_']
  sum = 0
  for value in dic_dlabel.values():
    sum = sum + float(value)
  mitja = sum / len(dic_dlabel.values())
  Grans = 0
  Petits = 0
  Mitjans = 0
  for value in dic_dlabel.values():
    if float(value) > mitja * 1.2:
      Grans = Grans + 1
    elif float(value) < mitja * 0.8:
      Petits = Petits + 1
    else:
      Mitjans = Mitjans + 1
  d_classificacio = {'Grans': Grans, 'Mitjans': Mitjans, 'Petits': Petits}


  return d_classificacio
```

```
In [11]: inferencia(img, model_path)
         Requirement already satisfied: skan in /usr/local/lib/python3.6/dist-packages (0.8)
Out[11]: {'Grans': 6, 'Mitjans': 7, 'Petits': 4}
```