Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

# TREBALL FINAL DE GRAU

**TÍTOL DEL TFG: Fully convolutional networks per segmentació d'imatges**

**TITULACIÓ: Grau en Enginyeria Telemàtica**

**AUTOR: Miguel Angel Archilla Álvarez**

**DIRECTOR: Esther Salamí San Juan**

**DATA: 7 de juliol del 2020**

**Títol:** Fully convolutional networks per segmentació d'imatges

**Autor:** Miguel Angel Archilla Álvarez

**Director:** Esther Salamí San Juan

**Data:** 07 de juliol del 2020

**Resum**

La tecnologia evoluciona molt ràpidament i en aquesta última dècada s'ha materialitzat el començament d'un futur que no sabem encara realment fins on arribarà.

La irrupció del machine learning porta al nostre dia a dia pensaments que es veien molt llunyans a començaments del segle XXI i obre reptes en llocs on mai s'havia imaginat.

L'avanç de la intel·ligència artificial ha permès que siguem capaços de desenvolupar maquines intel·ligents capaces d'aprendre i millorar per si soles, donant així l'aparició de projectes com el vehicle autònom, i cada dia millor, permet a un usuari realitzar un desplaçament on sols ha de supervisar les decisions del vehicle.

Així mateix, fent ús de l'aprenentatge profund, assolim en el món de la medicina el repte que amb només imatges mèdiques, crear una màquina capaç de descobrir si un pacient alberga una malaltia dins seu.

Plantejant escenaris com que en un futur la recerca i creació de nous medicaments vindrà liderada per processos d'intel·ligència artificial.

L'objectiu d'aquest treball és investigar les diferents tècniques d'anàlisis d'imatge, i en concret l'aprenentatge profund basat en segmentació semàntica, per avaluar la viabilitat de la implementació d'una CNN en un UAS per l'anàlisi de camps agrícoles amb Keras i Azure Machine Learning. Abordant problemes típics com la recerca i augment del conjunt de dades, elecció d'un entorn de treball i realització d'entrenaments i ajustament de paràmetres.

El resultat obtingut és positiu i amb ell es planteja diverses solucions, però existeix un gran marge de millora amb la recerca de noves tècniques d'implementació.

**Title:** Fully convolutional networks per segmentació d'imatges

**Author:** Miguel Angel Archilla Álvarez

**Director:** Esther Salamí San Juan

**Date:** July 7th 2020

## Overview

The technology has evolved very fast and in this last decade has started a future we don't know for sure where it will arrive.

The emergence of the machine learning has brought to our present days thoughts that were very far away at the beginning of the XXI century.

The progress of the artificial intelligence has allowed the development of new intelligent machines capable to learn and improve by themselves. With the help of this technology projects like the autonomous vehicle is continually improving, allowing a user do a displacement where only needs to supervise the vehicle decisions.

Moreover, using the deep learning we can design in the medicinal world a machine capable to discover a disease with only the help of a few medical images and thereby, save lives more efficiently.

Having in a future, scenarios where the research and the creation of new medicines will be led by artificial intelligence processes.

The objective of this project is to investigate the different techniques of image analysis, in more detail the deep learning based on semantic segmentation, to evaluate the implementation viability of a CNN in a UAS for the analysis of agricultural fields with Keras and Azure Machine Learning. Approaching typical problems as the research and the increase of the dataset, the choice of a work environment and the achievement of trainings and parameters adjustments.

The obtained result is positive, raising the idea of possible solutions, but there is a big improvement gap with the research of new implementation techniques.

# *Agraïments*

*M'agradaria expressar el meu agraïment primer de tot a la meva tutora Esther Salamí San Juan per aquesta oportunitat i el suport realitzat en tot el projecte.*

*Agrair a les professores i professors de l'EETAC l'ajuda i comprensió aportada en l'àmbit acadèmic que han fet possible compaginar l'àmbit laboral amb els estudis, sense ells no hauria sigut possible arribar fins aquest punt.*

*I per últim, el més sincer agraïment als familiars, amics i en especial a la meva parella, que han fet d'aquest propòsit una realitat.*

# ÍNDEX

# 1. Introducció i objectius

## 1.1. Introducció

L'any 1956 John McCarthy, persona molt reputada dins del camp d'intel·ligència artificial (IA), introdueix per primera vegada el terme "intel·ligència artificial" definint-la com "la ciència i l'enginy de crear màquines intel·ligents, especialment programes de computació intel·ligents." [1]

Ha estat a partir d'aquesta última dècada amb l'explosió de les noves tecnologies, i en concret amb el potencial de les GPUs i la computació en paral·lel juntament amb la consolidació d'internet, les xarxes socials i l'aparició del Big Data quan aquest terme agafa realment força i comença a fer-se realitat.

En l'actualitat vivim cada cop més rodejats de sistemes que prenen accions per nosaltres, aquests són capaços d'analitzar l'entorn que els envolta, aprendre i millorar amb l'experiència.

Un cas molt senzill és l'aplicació de reconeixement de veu, que implementa intel·ligència artificial per entendre'ns i dur a terme les accions que sol·licitem.

Altres casos d'èxit són el sector automobilístic amb l'aparició dels vehicles autònoms i la medicina implementant tècniques de visió per computació derivades de l'IA per la detecció precoç de malalties.

Les aplicacions i implementacions que ofereix la intel·ligència artificial són pràcticament infinites. Arran dels avanços i investigacions d'aquests últims anys en implementacions en sistemes de reconeixement d'imatges utilitzant UAV, neix la motivació de continuar la recerca de l'article *On-the-Fly Olive Tree Counting Using a UAS and Cloud Services* [2], on es fan servir tècniques tradicionals de processat d'imatge per detectar i comptar en temps real el número d'olivers que hi ha en una parcel·la a partir de les imatges preses per un dron, i realitzar un projecte per investigar i aprendre les diferents eines que ofereix l'IA dins del camp de la visió per computació per aconseguir crear una "màquina" capaç de segmentar imatges.

## 1.2. Objectius del treball

L'objectiu d'aquest projecte és estudiar les diferents tècniques d'aprenentatge profund basades en l'aprenentatge autònom per intentar desenvolupar un model predictiu que sigui capaç d'identificar en un camp de cultius d'olivers els següents objectes: oliver, herba i terra, amb l'objectiu de facilitar la tasca de recompte i d'anàlisis de la vegetació i el terreny.

Partint de la premissa de no comptar amb recursos econòmics per desenvolupar aquest projecte i de disposar d'un conjunt d'imatges molt reduït amb només 10 imatges del terreny, per aconseguir complir l'objectiu del projecte serà necessari:

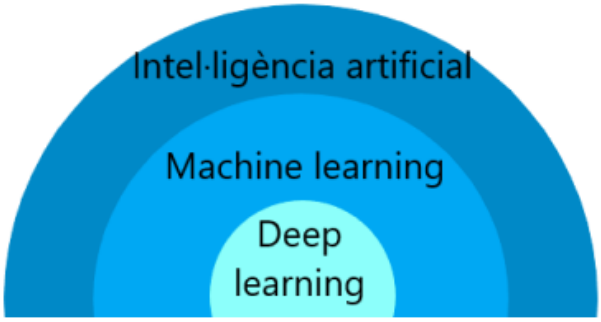
- Conèixer l'estat de l'art de la visió per computació
- Estudiar les diferents solucions que existeixen en l'actualitat.

- Estudiar els diferents llenguatges de programació i frameworks per poder implementar una solució.
- Preparar un entorn de treball que sigui capaç de suplir els requisits tècnics per dur a terme el projecte.
- Realitza i validar diferents propostes de solució.

# 2. Estat de l'art

## 2.1. Introducció a la intel·ligència artificial

Intel·ligència artificial, Machine learning i Deep learning són 3 conceptes molt semblants però que cal diferenciar correctament.




*Il·lustració 1 Esquema d'intel·ligència artificial*

### 2.1.1. Intel·ligència artificial

**Intel·ligència artificial (IA)** és fer que una màquina pensi, percebi i actuï com un ésser humà o inclús millor que aquest. És un objectiu molt lluny avui dia.

### 2.1.2. Machine learning

**Machine learning o aprenentatge automàtic (ML)** és un subconjunt de la IA que té com a objectiu que una màquina aprengui a fer una cosa sense que hagi sigut programada per aquella tasca.

Aplicant algoritmes que li permeten identificar patrons complexos, finalment una màquina serà capaç d'analitzar i aprendre de les dades.

Un senzill exemple seria el fet de voler detectar quines imatges contenen un gat, per aquesta tasca podríem aplicar dos solucions.

1. Desenvolupar un codi que sigui capaç de classificar la imatge com a gat quan detecta **una imatge concreta** amb un gat, però això només seria útil davant la mateixa imatge.
2. Entrenar una màquina fent ús de moltes imatges d'animals i correctament etiquetades, fent així que la màquina aprengui que hi ha imatges que es diuen «gat» i aquestes comparteixen entre elles característiques i patrons com per exemple un contorn d'una figura, la forma del nas, ulls, orelles, potes, etc.

La segona opció és la més versàtil perquè serà capaç de detectar si en una imatge existeix un gat, independentment del tipus d'imatge que sigui; això és la que denominem com machine learning.

### 2.1.3. Deep learning

**Deep learning o aprenentatge profund (DL)** és una tècnica concreta de l'aprenentatge autònom que es basa en l'anàlisi de dades fent **ús de xarxes neuronals artificials** que intenten imitar el cervell humà.

Aquestes xarxes estan compostes per diferents nivells de jerarquia, on inicialment el primer nivell aprèn una cosa senzilla i passa la informació al següent nivell que aprendrà una cosa més complexa, iterant així N vegades fins a obtenir el resultat esperat.

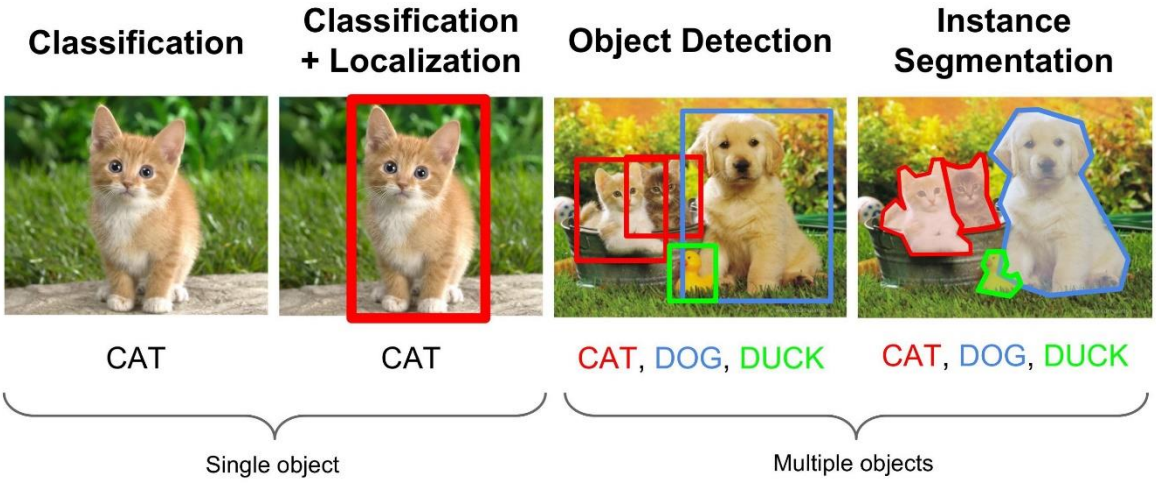
Seguint l'exemple anterior del gat, en aquest cas la xarxa neuronal analitzaria per primera vegada la imatge per diferenciar zones clares i fosques obtenint així el contorn de la figura, amb aquesta informació el següent nivell podria detectar els diferents atributs de la figura, després de N nivells més podria dir si els objectes simples són ulls, boca, nas o potes, fins al final poder combinar-los tots i decidir en l'últim nivell jeràrquic que allò analitzat és un gat.

## 2.2. Mètodes per classificar una imatge

A dia d'avui l'anàlisi i processament d'imatge és un camp molt important dins del món del machine learning  en camps com per exemple la medicina, l'automoció, la seguretat i l'agricultura.

En l'actualitat existeixen múltiples mètodes per reconèixer que conté una imatge:

- La **classificació** únicament identifica/etiqueta una imatge amb el seu objecte.
- La **classificació i localització** permet identificar i posicionar l'objecte dins de la imatge.
- La **detecció d'objectes** ens permet detectar els diferents objectes que té una imatge.




*Il·lustració 2 Tipus de classificació de una imatge* [3]

Per al nostre propòsit, el més important està relacionat amb la **segmentació** que es tracta d'una tècnica que detecta a quina categoria d'objecte pertany cada píxel de la imatge.


*Il·lustració 3 Segmentació per semàntica i segmentació per instancia* [4]

Dins de la segmentació d'imatges existeixen dos tipus:

- La **segmentació semàntica** és una tècnica capaç de detectar quin tipus d'objecte és cada píxel d'una imatge.
- La **segmentació per instàncies**, igual que la segmentació semàntica, pot detectar a quina classe pertany cada píxel, però a més a més també pot diferenciar per instància.

En l'exemple mostrat en la Il·lustració 3 es pot apreciar com la segmentació semàntica només detecta la categoria persona, mentre que la segmentació per instància a més és capaç de detectar els diferents nombres de persones que existeixen.

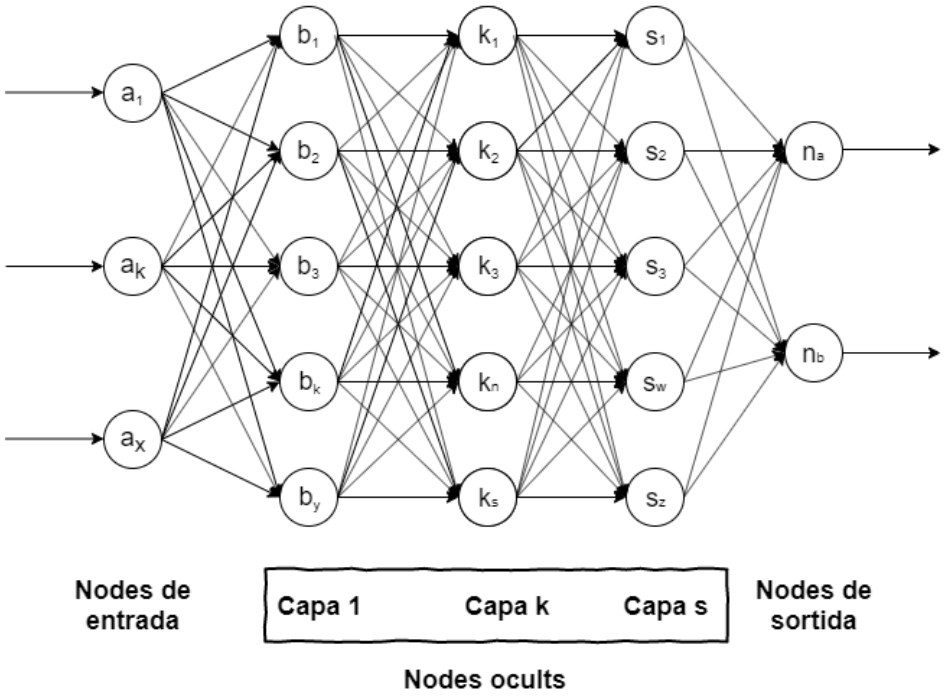
## 2.3. Xarxes neuronals convolucionals

Les xarxes neuronals convolucionals, en angles conegudes com a convolutional neuronal network (**CNN**), són les xarxes més populars dins del camp d'anàlisis d'imatge i vídeo donat els grans resultats que ofereixen.

### 2.3.1. Xarxes neuronals

Tal com es mostra a la Il·lustració 4 una xarxa neuronal està formada bàsicament per múltiples neurones artificials (**nodes**), que en conjunt conformen **capes** i alhora cada capa està interconnectada amb la següent per tal de poder enviar així la informació entre elles.

Totes les xarxes neuronals estan definides per una **capa d'entrada**, una **capa de sortida** i moltes **capes ocultes** intermèdies.




*Il·lustració 4 Diagrama d'una xarxa neuronal*

Aquestes xarxes aplicades a l'anàlisi d'imatge no són molt òptimes atès que una imatge a color de mida 100x100 conté 10.000 píxels i cada píxel conté 3 fragments de color (RGB) donant com a resultant 30.000 valors d'entrada. Si per exemple una de les capes ocultes conte 100 nodes, això implicarà 3.000.000 de connexions, nom massa gran que fa inviable la seva implementació.

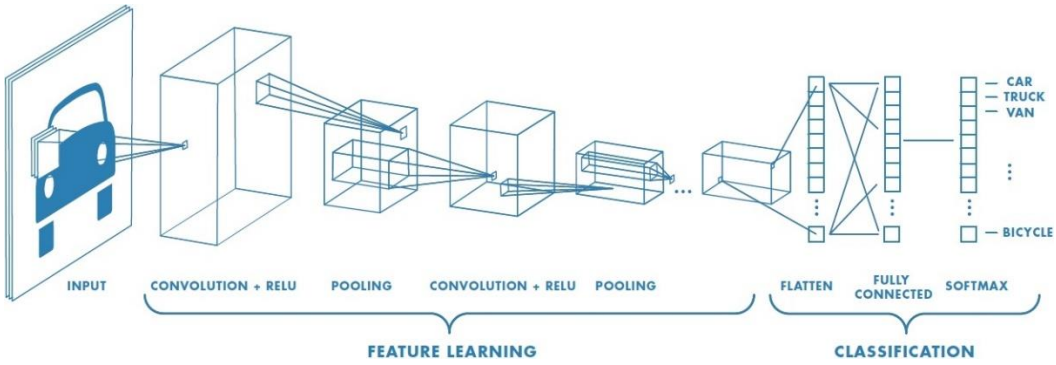
### 2.3.2. Xarxes neuronals convolucionals

Les xarxes neuronals convolucionals (CNN) són un tipus especial de xarxes neuronals capaces de capturar amb èxit les dependències espacials i temporals d'una imatge mitjançant filtres rellevants.

La característica principal d'una CNN és que permet reduir les imatges de tal forma que sigui més fàcil i ràpides de processar-les sense perdre en cap moment les propietats crítiques per a realitzar una bona predicció.

L'estructura d'una CNN es pot dividir en dos parts:

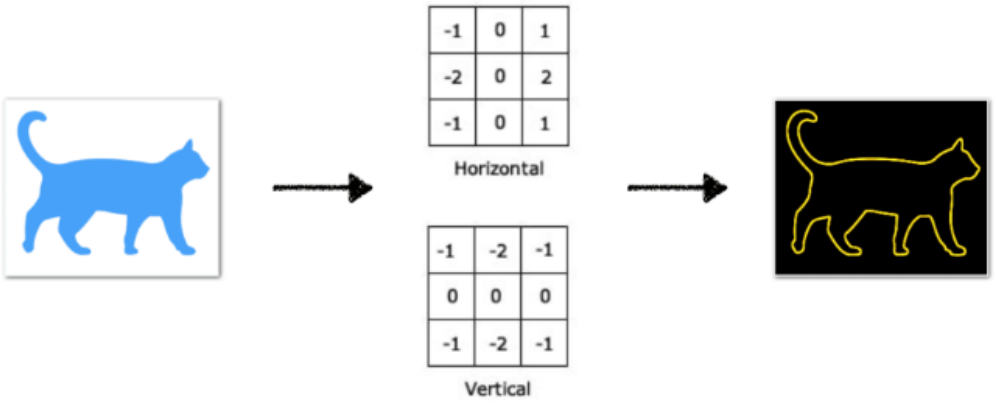
- **Aprenentatge de característiques:** és basa en implementar diferents filtres com la convolució i l'agrupació (pooling) per tal de depurar l'entrada i extreure així la informació primordial necessària.
- **Classificació:** és basa en tractar i entendre aquesta informació per tal de dur a terme una decisió.




*Il·lustració 5. Exemple d'una xarxa amb moltes capes de convolució* [5]

### *2.3.2.1. Aprenentatge de característiques*

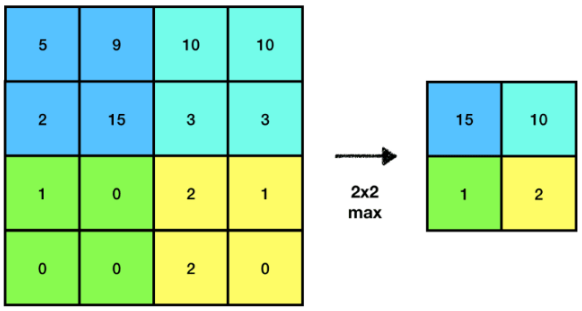
Donada una imatge d'entrada, amb una primera capa de convolució podem **extreure les característiques d'alt nivell** com per exemple el contorn de la figura.




*Il·lustració 6 Exemple de convolució per extreure característiques de alt nivell* [6]

Normalment després d'una o múltiples convolucions s'acostuma a fer ús de l'**agrupació** que té com a objectiu reduir la mida de l'entrada per així **disminuir la potència computacional requerida per al processament de dades**, també és útil per extreure característiques dominants que són invariants del posicionament i de la rotació.

Existeixen dos tipus d'agrupacions, **l'agrupació màxima** retorna el valor màxim de la part que cobreix el kernel, per contra **l'agrupació mitjana** retorna la mitjana de tots els valors de la part que cobreix el kernel.

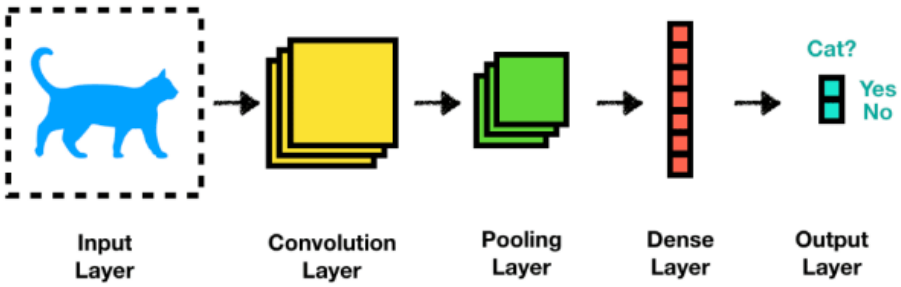



*Il·lustració 7. Agrupació màxima de 2x2 a una entrada de 4x4* [6]

A mesura que anem concatenant més capes de convolució tindrem fragments d'imatge més petits i podrem capturar les **característiques de baix nivell** com contorns, color, orientació del degradat, etc.

*2.3.2.2. Classificació*

Un cop analitzada la imatge i conegudes les seves característiques és hora d'avaluar aquesta informació i decidir (predir) que és allò que s'ha visualitzat.




*Il·lustració 8. Exemple simplificat del tractament de la informació en una CNN* [6]

Per aquesta tasca s'utilitzaran capes denses (**dense layers**), és a dir, les xarxes neuronals completament connectades ja conegudes en l'apartat 2.3.1 que connectaran cada neurona des de la capa anterior a la següent.

### 2.3.3. Arquitectura i models d'una xarxa neuronal convolucional

A nivell lògic, una CNN està definida per un **model** i una **arquitectura**, el model defineix les capes i les accions d'anàlisis disponibles, per contra l'arquitectura defineix a baix nivell com s'implementa/interactuen les capes i accions definides en el model.

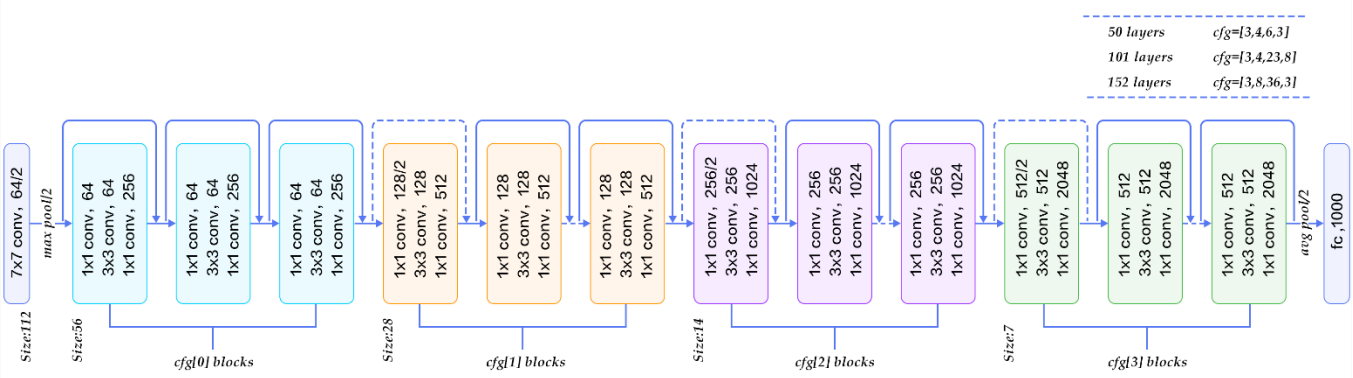
Per exemple quan nosaltres apliquem un entrenament d'una CNN, el model aplica les accions per extreure la informació, assignar pesos i finalment arribar realitzar una predicció. En aquest punt l'arquitectura pot modificar la connexió dels nodes de la xarxa neuronal alimentant un node amb informació d'una capa inicial o bé fent omissió de connexions amb l'objectiu de millorar el resultat en la predicció final.

### 2.3.4. Xarxes neuronals convolucionals existents

En l'actualitat existeixen diferents models per la segmentació semàntica; per al nostre objectiu ens centrarem en els següents models estàndards:

**ResNet** [7] és el model definit per Microsoft i va obtenir una precisió del 96,4% en la competició ILSVRC [8] (ImageNet Large Scale Visual Recognition Challenge) l'any 2016.

Es caracteritza per incrementar el nombre de capes introduint una connexió residual (amb una capa d'identitat).




*Il·lustració 9. Arquitectura del model ResNet* [9]

**VGG-16** [10] és el model definit per Oxford i va obtenir una precisió del 92,7% en la competició ILSVRC [8] l'any 2013.

En comparació amb ResNet, VGG-16 al ser més petita, està formada per 16 capes, és més ràpida d'entrenar però en conseqüència possiblement no oferirà una precisió molt elevada.

| ConvNet Configuration | | | | | |
|---|---|---|---|---|---|
| A | A-LRN | B | C | D | E |
| 11 weight layers | 11 weight layers | 13 weight layers | 16 weight layers | 16 weight layers | 19 weight layers |
| input (224 × 224 RGB image) | | | | | |
| conv3-64 | conv3-64<br>**LRN** | conv3-64<br>**conv3-64** | conv3-64<br>conv3-64 | conv3-64<br>conv3-64 | conv3-64<br>conv3-64 |
| maxpool | | | | | |
| conv3-128 | conv3-128 | conv3-128<br>**conv3-128** | conv3-128<br>conv3-128 | conv3-128<br>conv3-128 | conv3-128<br>conv3-128 |
| maxpool | | | | | |
| conv3-256<br>conv3-256 | conv3-256<br>conv3-256 | conv3-256<br>conv3-256 | conv3-256<br>conv3-256<br>**conv1-256** | conv3-256<br>conv3-256<br>**conv3-256** | conv3-256<br>conv3-256<br>conv3-256<br>**conv3-256** |
| maxpool | | | | | |
| conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512<br>**conv1-512** | conv3-512<br>conv3-512<br>**conv3-512** | conv3-512<br>conv3-512<br>conv3-512<br>**conv3-512** |
| maxpool | | | | | |
| conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512<br>**conv1-512** | conv3-512<br>conv3-512<br>**conv3-512** | conv3-512<br>conv3-512<br>conv3-512<br>**conv3-512** |
| maxpool | | | | | |
| FC-4096 | | | | | |
| FC-4096 | | | | | |
| FC-1000 | | | | | |
| soft-max | | | | | |

*Il·lustració 10. Arquitectura del model VGG* [10]

**MobileNet** [11] és el model definit per Google dissenyat per ser implementat en dispositius mòbils. Està basada en una arquitectura optimitzada que utilitza convolucions separables en profunditat per a poder construir xarxes profundes i lleugeres.

Fent ús de dos hiperparàmetres globals simples que s'intercanvien entre latència i precisió, la xarxa és capaç d'ajustar-se i escollir la mida correcta per la seva aplicació en funció de les restriccions del problema.

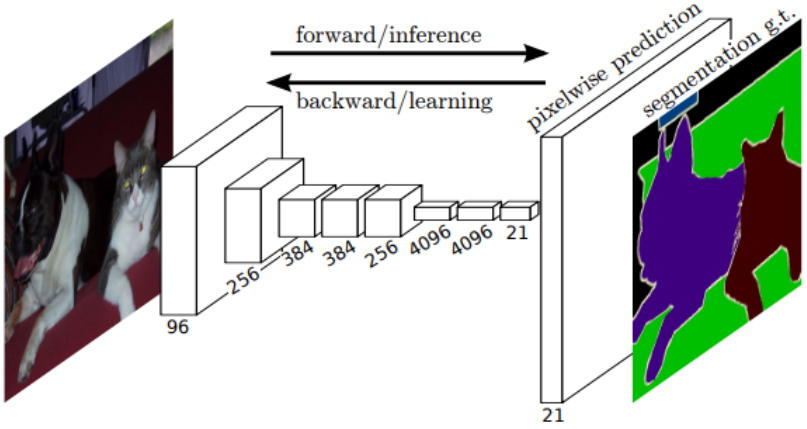
| Type / Stride | Filter Shape | Input Size |
|---|---|---|
| Conv / s2 | $3 \times 3 \times 3 \times 32$ | $224 \times 224 \times 3$ |
| Conv dw / s1 | $3 \times 3 \times 32$ dw | $112 \times 112 \times 32$ |
| Conv / s1 | $1 \times 1 \times 32 \times 64$ | $112 \times 112 \times 32$ |
| Conv dw / s2 | $3 \times 3 \times 64$ dw | $112 \times 112 \times 64$ |
| Conv / s1 | $1 \times 1 \times 64 \times 128$ | $56 \times 56 \times 64$ |
| Conv dw / s1 | $3 \times 3 \times 128$ dw | $56 \times 56 \times 128$ |
| Conv / s1 | $1 \times 1 \times 128 \times 128$ | $56 \times 56 \times 128$ |
| Conv dw / s2 | $3 \times 3 \times 128$ dw | $56 \times 56 \times 128$ |
| Conv / s1 | $1 \times 1 \times 128 \times 256$ | $28 \times 28 \times 128$ |
| Conv dw / s1 | $3 \times 3 \times 256$ dw | $28 \times 28 \times 256$ |
| Conv / s1 | $1 \times 1 \times 256 \times 256$ | $28 \times 28 \times 256$ |
| Conv dw / s2 | $3 \times 3 \times 256$ dw | $28 \times 28 \times 256$ |
| Conv / s1 | $1 \times 1 \times 256 \times 512$ | $14 \times 14 \times 256$ |
| $5\times$ Conv dw / s1 | $3 \times 3 \times 512$ dw | $14 \times 14 \times 512$ |
| $5\times$ Conv / s1 | $1 \times 1 \times 512 \times 512$ | $14 \times 14 \times 512$ |
| Conv dw / s2 | $3 \times 3 \times 512$ dw | $14 \times 14 \times 512$ |
| Conv / s1 | $1 \times 1 \times 512 \times 1024$ | $7 \times 7 \times 512$ |
| Conv dw / s2 | $3 \times 3 \times 1024$ dw | $7 \times 7 \times 1024$ |
| Conv / s1 | $1 \times 1 \times 1024 \times 1024$ | $7 \times 7 \times 1024$ |
| Avg Pool / s1 | Pool $7 \times 7$ | $7 \times 7 \times 1024$ |
| FC / s1 | $1024 \times 1000$ | $1 \times 1 \times 1024$ |
| Softmax / s1 | Classifier | $1 \times 1 \times 1000$ |

*Il·lustració 11. Arquitectura del model MobileNet* [11]

Un cop seleccionats els models hem de definir l'arquitectura de segmentació que implementarem, aquestes vindran definides pel nombre d'imatges d'entrenament, la mida de les imatges i la naturalesa de les imatges.

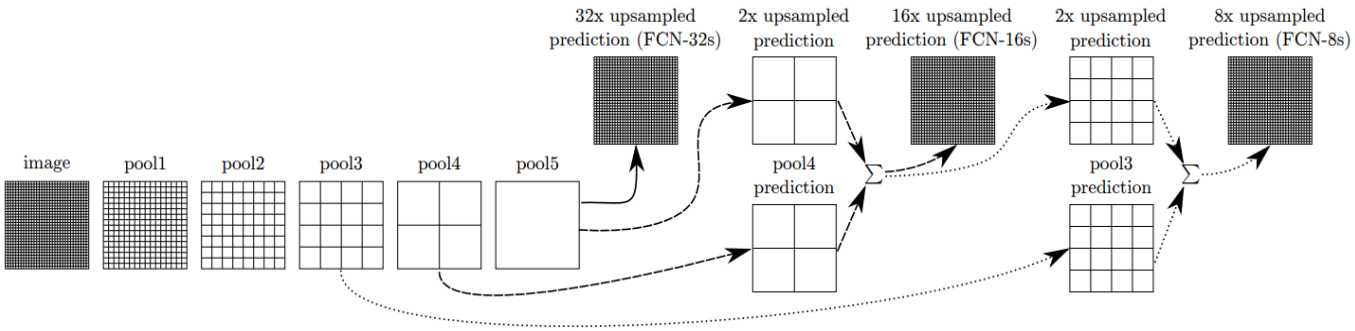
**FCN** [12] és una de les primeres arquitectures de segmentació basada en capes convolucionals completament connectades. La xarxa està formada per dues parts:

- Camí de mostreig ascendent (forward/inference) utilitzat per capturar la informació semàntica, és a dir, per extreure e interpretar el context del píxel/fragment d'imatge.
- Camí de mostreig descendent (barckward/learning) utilitzat per recuperar la informació espacial i així localitzar la posició del píxel/fragment d'imatge dins de la imatge.



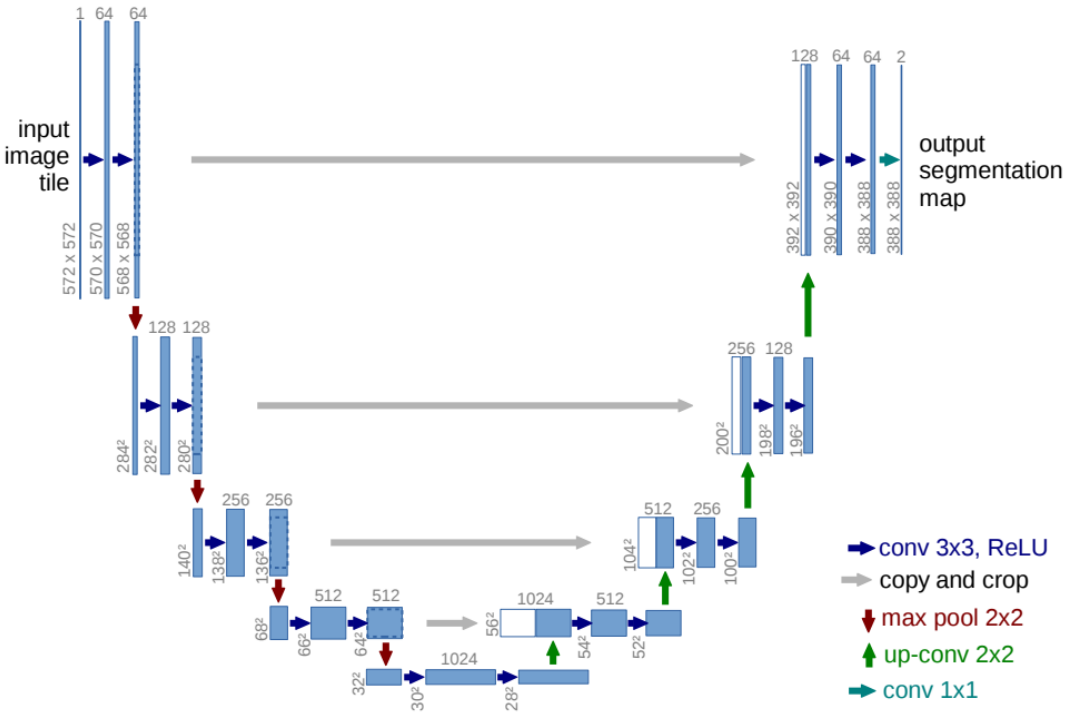

*Il·lustració 12. Arquitectura FCN* [12]

En l'actualitat existeixen diferents variants: FCN-8, FCN-16 i FCN-32 i es diferencien entre elles en la precisió espacial obtinguda  a la sortida, donat que la predicció es realitza per cada una d'elles en moments diferents.




*Il·lustració 13. Gràfic acíclic dirigit de la predicció d'una FCN* [12]

**U-Net** [13] ha sigut dissenyada sobre FCN, fa ús d'una estructura codificador-descodificador simètric amb connexions d'omissió entre les capes de codificació i descodificació i és molt utilitzada en entorns mèdics i entorns amb poques dades, ja que és capaç d'obtenir bons resultats amb menys sets d'entrenament.

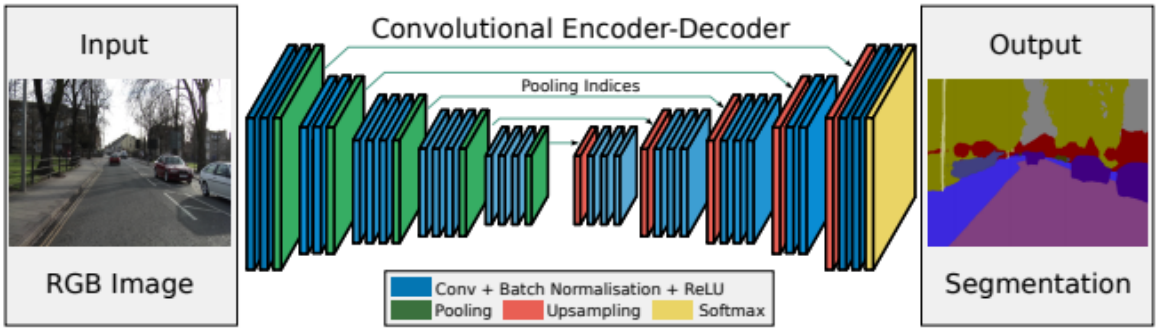



*Il·lustració 14. Arquitectura U-net* [13]

La xarxa codificadora processa la imatge fins a transformar-la en un vector per obtenir el context i característiques de la imatge, després la xarxa descodificadora que consisteix en

capes desconvolucionals reconstruirà la imatge per obtenir la localització, durant aquest procés de descodificació la xarxa utilitzarà les connexions d'omissió per ajudar al generador a reconstruir la imatge més precisa.

**SegNet** [14] igual que U-Net també fa ús d'una estructura codificador-descodificador amb capes simètriques. És diferència de la resta perquè la xarxa descodificadora fa ús dels índexs d'agrupació màxims generats a la xarxa codificadora i no fa ús de connexions d'omissió.




*Il·lustració 15. Arquitectura SegNet* [14]

Existeixen altres arquitectures com DeepLab i PSPNet però aquestes estan enfocades a tractar imatges de grans dimensions (500x500) amb una gran quantitat de classes dins l'escena.

Per a conjunts de dades simples, i una petita quantitat d'objectes, els models simples com FCN o Segnet podrien ser suficients.

### 2.3.5. Llenguatge de programació, llibreries i frameworks

Actualment podem desenvolupar un projecte de IA amb diferents llenguatges de programació com C++, Java, R, entre altres, però el més popular i més utilitzat avui dia és Python a causa de la simplicitat i facilitat a l'hora de treballar amb ell.

Per altra banda, existeixen multitud de frameworks per dur a terme tasques de IA com Tensorflow desenvolupat per Google, PyTorch desenvolupat per Facebook, MathWorks desenvolupat per MatLab, CNTK desenvolupat per Microsoft entre altres més.

L'elecció d'un llenguatge o eina de treball en molts casos estarà condicionada per la seva facilitat d'ús, la documentació disponible, la seva comunitat, el seu rendiment en entorns productius, casos d'ús per companyies externes, si és de pagament o bé gratuïta, etc.

# 3. Metodologia

Els experiments estaran basats en **aprenentatge supervisat** on els algoritmes treballaran amb dades etiquetades que podran consultar al final de la predicció per conèixer quant d'acurat ha estat el resultat, millorant el resultat amb el pas del temps a base de prova, error i correcció.

Tots els experiments es realitzaran amb **Keras** [15] que és una API de deep learning escrita en **Python** i que implementa la famosa llibreria **TensorFlow** [16] desenvolupada per Google.

Per dur a terme els diferents experiments utilitzarem la llibreria **keras-segmentation** [17] per fer ús dels models bàsics predefinits sense cap classe d'entrenament previ.

La decisió d'utilitzar aquestes eines es basa en la facilitat que ofereixen a l'hora de desenvolupar una aplicació de deep learning per primer cop i en l'ampli suport de la comunitat.

Finalment s'ha decidit analitzaran l'eficàcia dels models **VGG-16, ResNet50 i MobileNet** sobre les arquitectures **FCN-8 i UNET** per a diferents mides d'imatges.

Aquesta elecció ve motivada pels bons resultats que han obtingut en altres tipus de proves, la facilitat d'entrenament amb conjunts de dades petits i per la idea d'analitzar models de diferents mides aplicables en diferents entorns.

## 3.1. Elaboració del conjunt de dades

Una part molt important per ser capaç de dur a terme qualsevol experiment amb ML és disposar d'un **conjunt d'imatges prou gran i correctament etiquetades** per poder executar els models i obtenir bons resultats. La realitat és que molts cops o hi ha poques dades disponibles o bé aquestes no estan etiquetades, cosa que fa que gran part del temps estigui dedicat a aquesta tasca i és important conèixer tècniques i automatitzar processos per intentar reduir aquest temps al màxim.

En el nostre cas hem partit de la premissa de només disposar de 10 imatges d'alta resolució (3808x2912 píxels) sense etiquetar, per la qual cosa serà necessari etiquetar-les i incrementar el nombre de mostres fent ús de tècniques **d'augment de dades**.

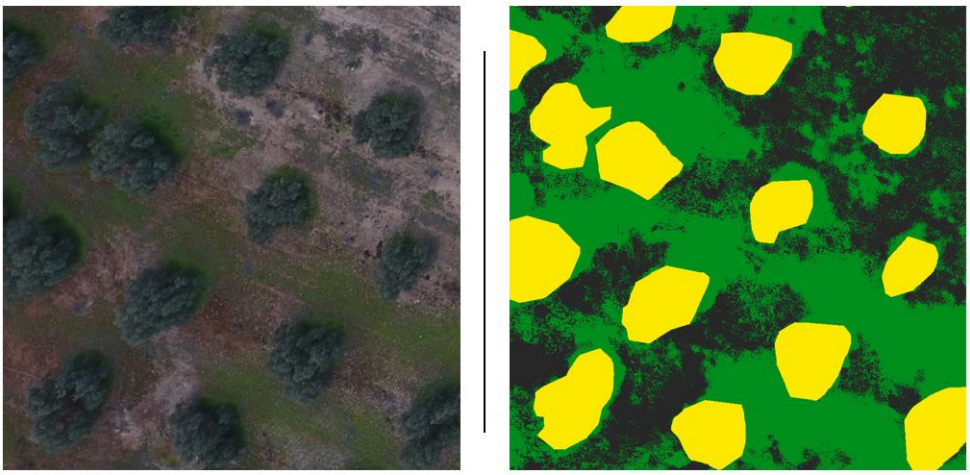
Aquestes 10 imatges pertanyen a la investigació realitzada a l'article *On-the-Fly Olive Tree Counting Using a UAS and Cloud Services* [2] i han sigut cedides per la realització d'aquest projecte.

*Il·lustració 16. Imatge d'exemple del nostre conjunt de dades*

Per tal d'etiquetar les imatges existeixen diferents eines en línia gratuïtes com per exemple **LabelMe** [18] que ens permet generar i gestionar el conjunt de dades (**dataset**) de forma manual o bé serveis de pagament més sofisticats com **Amazon SageMaker Ground Truth** [19] que ens permet delegar aquesta tasca perquè sigui realitzada per un procés de IA o per proveïdors externs.

Per al nostre cas en tractar-se d'imatges amb moltes petites zones d'un tipus, com per exemple molts petits fragments de part del terreny amb herba i d'altres amb terra, vam decidir fer ús de l'editor d'imatge de codi lliure **GIMP** [20] perquè disposa d'eines per a seleccionar múltiples elements semblants basats en el color.

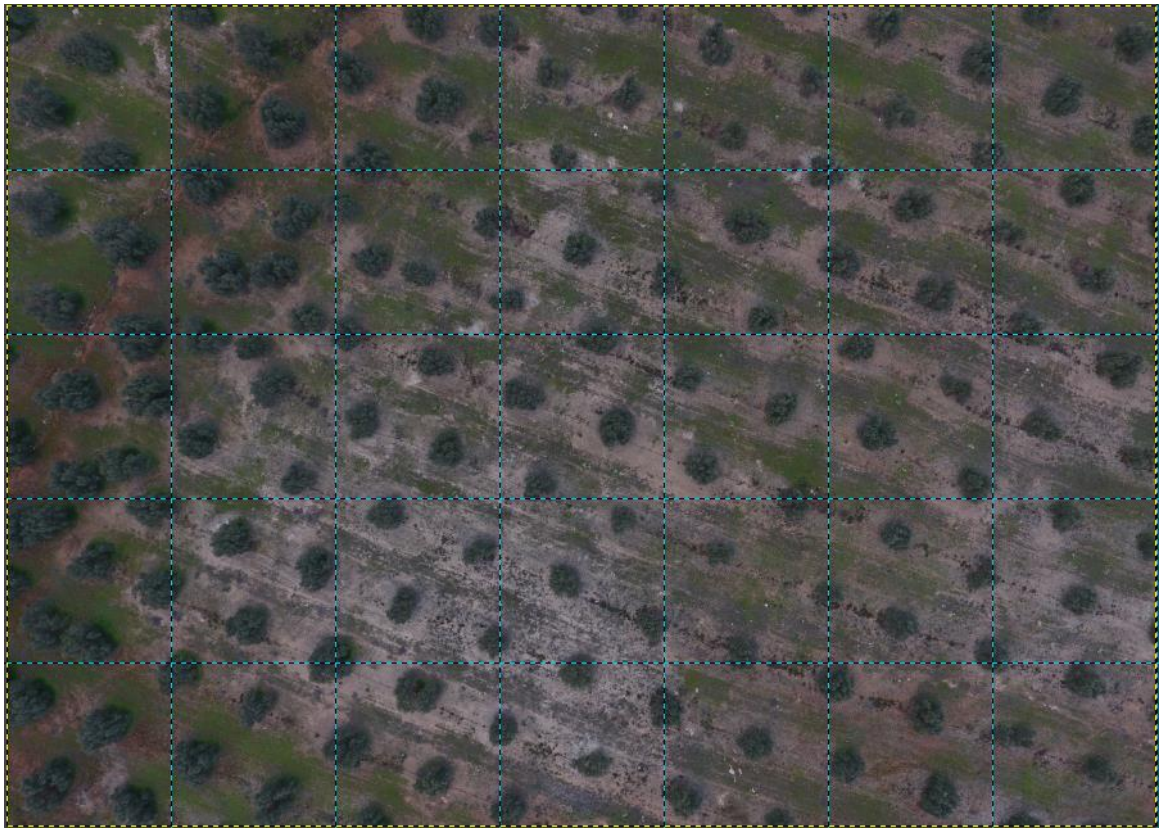
A la Il·lustració 17 podem veure un exemple d'un fragment d'imatge original i de la imatge de referència com a predicció vàlida (**ground truth**) que hem generat on cada color representa una classe diferent, en aquest cas apreciem 3 colors que fan referència a l'arbre, l'herba i el terra.



*Il·lustració 17. Fragment d'imatge original i ground truth generats*

Un cop etiquetades les 10 imatges, serà necessari aplicar diferents tècniques per poder augmentar el nombre d'imatges disponibles per a l'experiment, en el nostre cas hem incrementat el nombre de mostres de la següent manera:

1. Dividint la imatge principal en mostres més petites.



*Il·lustració 18. Mostreig d'una imatge original en 35 submostres*

2. Aplicant girs horitzontals, verticals i modificacions aleatòries (dins d'un rang acotat) en la il·luminació i el contrast.

Per aplicar aquesta tècnica es va generar un script per automatitzar aquest procés [21].

*Il·lustració 19. Exemple de un fragment amb diferents alteracions*

Aplicant aquest tipus de tècniques hem aconseguit passar de tenir 10 mostres a 1400 mostres.

$$10\ imatges\ originals \cdot 35\ imatges/imatges\ originals\ = 350\ imatges$$

$$350\ imatges \cdot\ 4\ alteracions\ diferents/imatge\ = 1400\ imatges$$

*Equació 1. Càlcul de l'augment de dades*

Es van explorar altres tècniques visuals per modificar el conjunt de dades fent ús d'eines existents a la llibreria Keras, com es mostra en el següent exemple [22], però en el nostre cas no va resultar el més apropiat a causa que el ground truth resultant era una imatge d'escala de grisos i això entrava en conflicte amb el codi utilitzat més endavant. Tot i això en altres circumstàncies amb les correccions adequades en el codi farien d'aquesta una opció molt valida a tenir en compte.

És recomanable fer ús de l'augment de dades donat que s'ha demostrat que beneficia la creació dels models predictius, accelerant la seva convergència i evitant el sobreajustament (**overfitting**) fent que aquests generalitzin millor i siguin igual d'òptims amb imatges diferents de les utilitzades en el procés d'entrenament [23].

## 3.2. Entorns d'execució

La correcta decisió d'un entorn d'execució per entrenar el model predictiu serà clau per poder dur a terme l'entrenament en el menor temps possible. També serà molt important optimitzar al màxim els recursos disponibles per tal de reduir al mínim el cost econòmic.

### 3.2.1. Anàlisis dels diferents entorns

#### *3.2.1.1. Entorn local*

Donat el gran volum i el tipus de dades utilitzades fan que aquest experiment no sigui adient realitzar-lo en un entorn local comú donat que amb els ordinadors convencionals no disposem dels recursos necessaris per realitzar entrenaments que durin menys d'1 dia [24] i això és a causa que el hardware de les GPUs és més adient davant el tipus d'operacions computacionals que duen a terme els processos de ML enfocats a visió per computació.

Tot i això, encara que disposéssim dels recursos econòmics suficients per adquirir aquest hardware seria convenient valorar quin ús real faríem, ja que són recursos molt cars (+2.000 €) i existeixen alternatives més econòmiques com el lloguer d'infraestructura en el núvol, on només paguem quan tenim el recurs reservat.

#### *3.2.1.2. Google Colaboratory & Google Cloud*

**Google Colaboratory** [25] és una eina molt útil per a començar a desenvolupar processos senzills de deep learning o simplement executar codi Python, ja que l'únic que necessitem és un navegador web, no requereix configuració prèvia, ens ofereix accés gratuït a GPUs i podem fer execucions sobre arquitectures **TPU** [26]. Per altra part, té una molt fàcil connexió amb Google Drive, pel que emmagatzemar i utilitzar el nostre dataset serà un problema.

El major inconvenient que té Google Colab. és que **no garanteixen l'ús ininterromput dels recursos**, fent així que no sigui una eina adequada per a execucions llargues, com és el nostre cas. Per altra banda, tampoc garanteixen l'ús del mateix model de GPU en cada execució, fet que ocasiona una incertesa amb el temps requerit en l'execució.

Existeix una versió de pagament molt econòmica anomenada **Google Colab. Pro,** on es millora l'accés habilitant GPUs de major rendiment i incrementen el temps de servei ininterromput (sense garanties), però de moment aquest servei sols està disponible a EEUU.

Es va explorar l'alternativa de connectar el nostre entorn de Google Colab. a un recurs dedicat de ML allotjat al servei de **Google Cloud** fent ús del crèdit gratuït que ofereixen als nous usuaris, però ràpidament es va descartar donat que no ens permeten en l'actualitat fer ús de màquines GPU amb crèdit gratuït.

#### *3.2.1.3. Azure Machine Learning*

Entre tots els serveis que ofereix Microsoft dins del portal de **Microsoft Azure** existeix un servei de pagament dedicat exclusivament a l'aprenentatge autònom anomenat **Azure Machine Learning**.

Azure ML disposa d'un entorn de treball molt extens amb el que podrem d'ur a terme totes les tasqués involucrades dins d'un projecte de ML com l'etiquetatge de dades, l'emmagatzemament de les dades, la creació i execució d'experiments, l'anàlisis dels resultats, etc.

En comparació amb la resta dels serveis prèviament mencionats considerem que és **l'eina més difícil** a l'hora d'iniciar-se, però un cop superada aquesta barrera d'iniciació, és **l'eina més completa** per a treballar tant individualment com amb equip.

Un punt molt positiu de Azure ML és que disposa de màquines virtuals GPUs ja predefinides les quals estan preparades per poder fer ús de **CUDA** [27] i així optimitzar al màxim els recursos i disminuir el temps d'entrenament. Actualment Azure ML permet fer ús d'alguns recursos GPUs amb crèdit gratuït.

### 3.2.2. El nostre entorn d'execució

Un cop analitzades diferents opcions, tenint en compte els recursos econòmics limitats i que els nostres experiments es caracteritzaran per requerir una important càrrega de treball, vam decidir fer ús de **Azure Machine Learning** pel fet de poder disposar de màquines GPUs amb crèdit gratuït i de tractar-se de l'eina més completa.

## 3.3. Entrenament

### 3.3.1. Conjunt de dades

Abans d'entrenar els diferents models necessitarem separar les imatges en diferents conjunts de dades, nosaltres hem elegit la següent distribució:

| Conjunt de dades per l'entrenament | | Conjunt de dades pel testeig |
|---|---|---|
| **TRAIN** | **VALIDATION** | **TEST** |
| 80% | 10% | 10% |

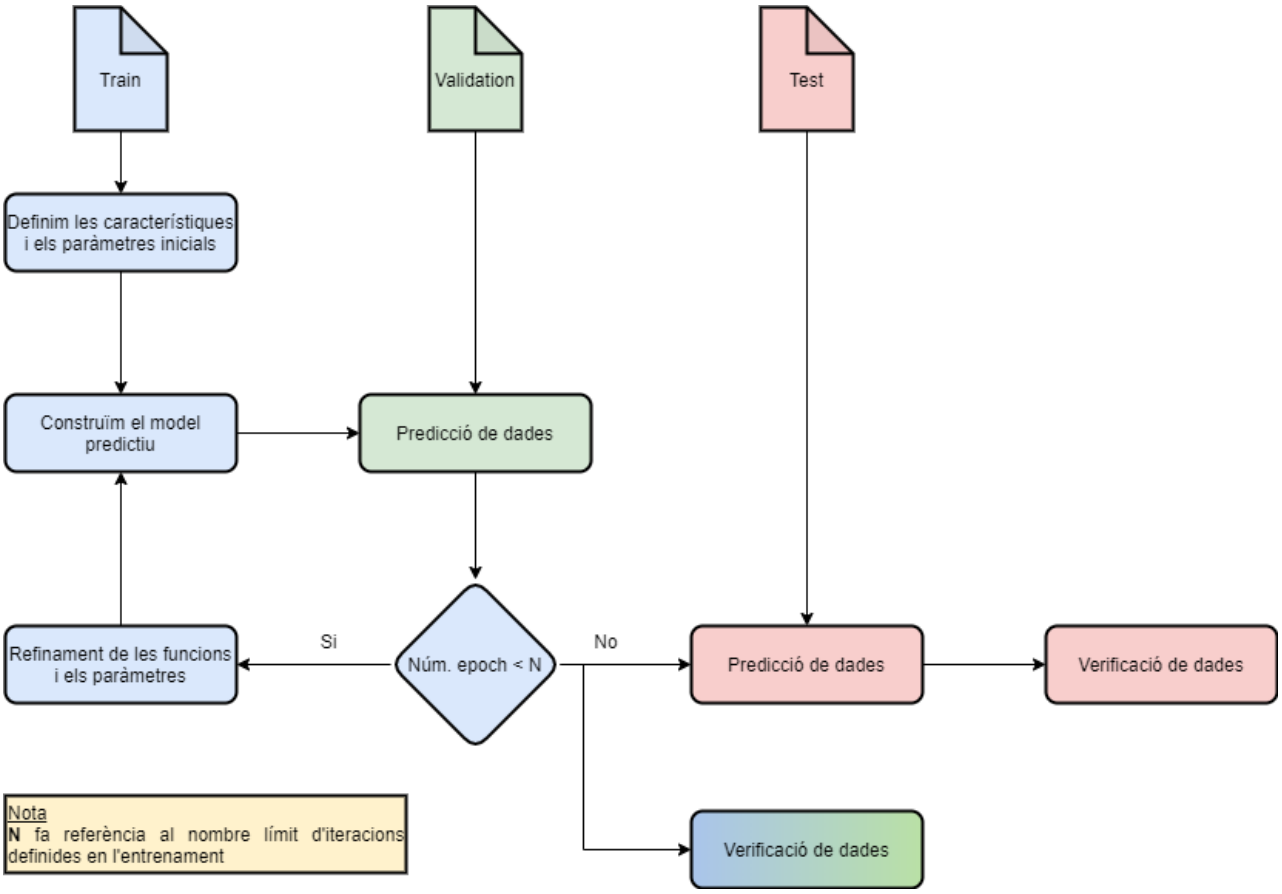
*Taula 1. Percentatges de distribució de les dades*

Cadascuna de les dades només podrà estar en un únic conjunt de dades.

El conjunt de dades **train** serà **utilitzat per entrenar el model de dades**, a cada iteració (**epoch**) el model analitzarà les imatges per aprendre les diferents característiques i intentarà predir els objectes que apareixen; validant amb la imatge resultant de referència serà capaç de saber com de bé ho ha fet i **d'autoajustar-se** per tal d'intentar millorar la següent iteració.

Alhora, sempre que és finalitza una iteració s'utilitza el conjunt de dades **validation** per **validar com de precís és en aquell moment el model**. A diferència del conjunt train, aquestes dades mai han sigut conegudes ni utilitzades per entrenar el model fent que sigui un valor resultant vàlid.

Un cop finalitzat l'entrenament, passarem a **verificar la precisió del model** simulant un cas d'ús real, és a dir, realitzant prediccions amb el conjunt de dades de **test**.




*Il·lustració 20. Flux de treball de l'entrenament d'un model*

### 3.3.2. Configuració de l'entrenament

Per defecte tots els models predictius seran entrenats fent ús de **l'optimitzador adadelta** [28] i una **funció de pèrdua d'entropia creuada categòrica** que en conjunt s'encarregaran d'elegir els paràmetres i pesos més òptims perquè la xarxa segueixi aprenent i millorant a cada iteració.

Per tal de poder dur a terme els diferents experiments serà necessari preparar el nostre entorn d'execució tal com s'explica en **l'Annex A - Configuració d'un entorn Azure Machine Learning**.

Un cop configurat l'entorn de treball haurem de definir el nostre experiment tal com s'explica en l'**Annex B - Script de l'experiment**.

Per tal de realitzar entrenaments òptims necessitarem definir correctament aquestes 5 variables:

- **epochs** defineix quants cops pot el model iterar el conjunt de dades per aprendre i millorar, per defecte tots els entrenaments seran executats amb 100 èpoques.
- **batch_size** indica quantes imatges podem passar-li a la xarxa de manera simultània, a més imatges més ràpid seran els entrenaments. Haurem de tindre en compte el nombre de capes que té el nostre model així com la memòria de la GPU a l'hora d'assignar aquest valor.

- **steps_per_epoch** indica quants blocs de batch_size executarem en una època, per arribar a utilitzar totes les dades en el nostre entrenament, hauríem de complir la següent regla:

$$batch\_size \cdot steps\_per\_epoch = número\ d'imatges\ en\ el\ dataset$$

*Equació 2. Relació del nombre d'imatges del dataset executades en una època*

- **val_batch_size** és el batch_size aplicat al dataset de validació
- **val_steps_per_epoch** és el step_per_epoch aplicat al dataset de validació

Aquests valors canviaran en funció del model i de la mida de les imatges del nostre dataset i no existeix una regla universal per calcular aquests paràmetres. En el nostre cas aquests valors els hem obtingut amb tècniques de hiperparàmetres tal com es mostra en l'apartat B.3.1 dels annexos.

És important assegurar que estem utilitzant correctament Tensorflow en mode GPU amb CUDA, tal com es mostra en l'apartat B.2.9 dels annexos, per tal de poder realitzar entrenaments més ràpids.

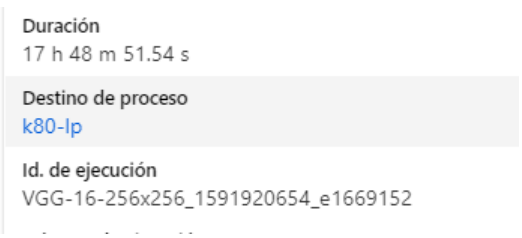
A les següents il·lustracions es pot apreciar com una mala implementació de CUDA a l'entrenament ocasionar èpoques més llargues i al final un entrenament més llarg.

```
138  Using TensorFlow backend.
139  2020-06-12 00:19:21.675136: W tensorflow/stream_executor/platform/default/dso_loader.cc:55] Could not load dynamic library 'libcuda.so.1'; dlerror: libcuda.so.1
140  2020-06-12 00:19:21.675205: E tensorflow/stream_executor/cuda/cuda_driver.cc:313] failed call to cuInit: UNKNOWN ERROR (303)
141  2020-06-12 00:19:21.675289: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:169] retrieving CUDA diagnostic information for host: a6965b565518406cb7bb2caee2
142  2020-06-12 00:19:21.675342: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:176] hostname: a6965b565518406cb7bb2caee26941e6000000
143  2020-06-12 00:19:21.675452: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:200] libcuda reported version is: Not found: was unable to find libcuda.so DSO l
144  2020-06-12 00:19:21.675539: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:204] kernel reported version is: 440.33.1
```

*Il·lustració 21. Execució de Tensorflow sense CUDA*

```
1/5 [=====>........................] - ETA: 8:13 - loss: 0.5288 - accuracy: 0.7780
2/5 [===========>..................] - ETA: 6:09 - loss: 0.5441 - accuracy: 0.7764
3/5 [=================>............] - ETA: 4:06 - loss: 0.5502 - accuracy: 0.7692
4/5 [======================>......] - ETA: 2:03 - loss: 0.5509 - accuracy: 0.7712
5/5 [==============================] - 635s 127s/step - loss: 0.5437 - accuracy: 0.7740 - val_loss: 0.6428 - val_accuracy: 0.7453
saved  outputs/checkpoint/vgg_unet_1.28
Epoch 30/100
```

*Il·lustració 22. Exemple del temps d'execució d'una època per un entrenament sense CUDA*




*Il·lustració 23. Temps d'execució final d'un entrenament sense CUDA*

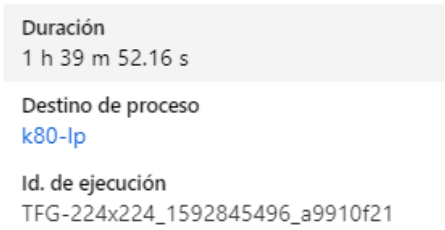
Per contra, a les següents il·lustracions es pot apreciar com el mateix entrenament implementant correctament amb CUDA ha trigat 12 vegades menys en executar-se.

```
Using TensorFlow backend.
/azureml-envs/azureml_0c953dcab8998ea39fb0ebcd6d817f0b/lib/python3.6/site-packages/tensorflow/python/framework/dtypes.py:526:
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
/azureml-envs/azureml_0c953dcab8998ea39fb0ebcd6d817f0b/lib/python3.6/site-packages/tensorflow/python/framework/dtypes.py:527:
…
2020-06-22 17:14:53.460924: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2 FMA
2020-06-22 17:14:53.596288: I tensorflow/compiler/xla/service/service.cc:150] XLA service 0x55f8d65aa980 executing computations on platform CUDA. Devices:
2020-06-22 17:14:53.596381: I tensorflow/compiler/xla/service/service.cc:158]   StreamExecutor device (0): Tesla K80, Compute Capability 3.7
```

*Il·lustració 24.Execució de Tensorflow amb CUDA*

```
 1/14 [=>............................] - ETA: 48s - loss: 0.6154 - accuracy: 0.7408
 2/14 [===>..........................] - ETA: 44s - loss: 0.6421 - accuracy: 0.7272
 3/14 [=====>........................] - ETA: 41s - loss: 0.6433 - accuracy: 0.7258
 4/14 [=======>......................] - ETA: 37s - loss: 0.6470 - accuracy: 0.7227
 5/14 [=========>....................] - ETA: 33s - loss: 0.6514 - accuracy: 0.7193
 6/14 [===========>..................] - ETA: 29s - loss: 0.6519 - accuracy: 0.7193
 7/14 [==============>...............] - ETA: 26s - loss: 0.6556 - accuracy: 0.7174
 8/14 [================>.............] - ETA: 22s - loss: 0.6562 - accuracy: 0.7177
 9/14 [==================>...........] - ETA: 18s - loss: 0.6565 - accuracy: 0.7178
10/14 [====================>.........] - ETA: 14s - loss: 0.6563 - accuracy: 0.7179
11/14 [======================>.......] - ETA: 11s - loss: 0.6540 - accuracy: 0.7190
12/14 [========================>.....] - ETA: 7s - loss: 0.6553 - accuracy: 0.7188
13/14 [==========================>...] - ETA: 3s - loss: 0.6509 - accuracy: 0.7209
14/14 [==============================] - 55s 4s/step - loss: 0.6497 - accuracy: 0.7216 - val_loss: 0.5912 - val_accuracy: 0.7695
saved  outputs/checkpoint/VGG-16_1.28
Epoch 30/100
```

*Il·lustració 25. Exemple del temps d'execució d'una època per un entrenament amb CUDA*




*Il·lustració 26.Temps d'execució final d'un entrenament amb CUDA*

Finalment, tal com s'explica a l'apartat B.3.2 dels annexos, per executar els diferents experiments es recomanable utilitzar **tècniques de parada preventiva** per aturar aquells entrenaments que donen un resultat molt dolent d'inici i que és preveu que no finalitzaran amb èxit.

### 3.3.3. Paràmetres de mesura

#### *3.3.3.1. Paràmetres de mesura en l'entrenament*

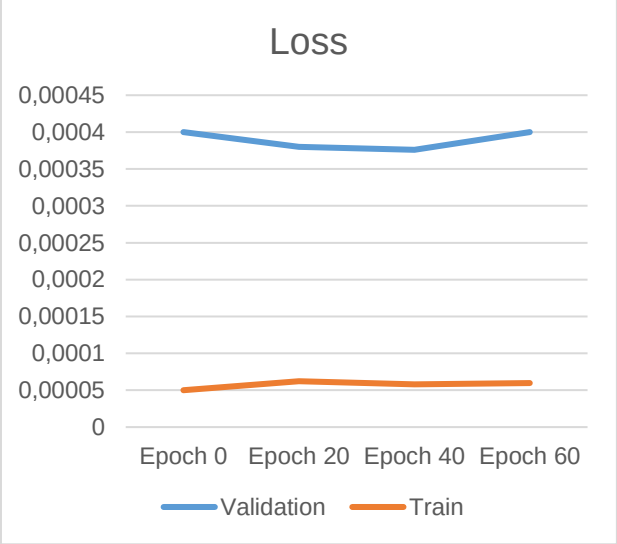
Per tal de poder validar l'èxit del nostre entrenament tindrem en compte les **mètriques de precisió de les prediccions** i les **mètriques de pèrdua**.

Com ja hem comentat anteriorment durant l'entrenament s'utilitzen els datasets train i validation sobre els que es realitzen prediccions a cada època, si els resultats de precisió

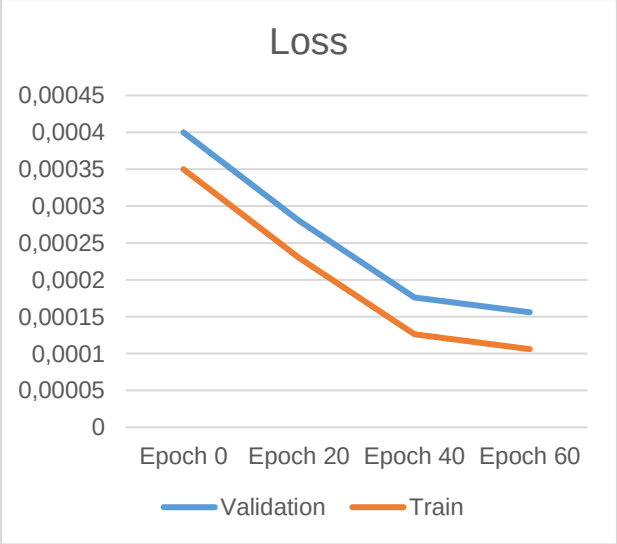
obtinguts entre ambos conjunts són semblants (com a màxim un **15%** de diferència entre si), podrem intuir que el resultat de l'entrenament ha sigut vàlid; això no significarà que la predicció sigui satisfactòria.

Per contra, si el resultat obtingut amb validation està molt per sota que el de train, indicarà que el model no generalitza correctament i haurem de revisar els paràmetres definits i repetir l'entrenament.

La mètrica de pèrdua també ens ajudarà a conèixer si el nostre model està correctament entrenat. Direm que un model no és capaç d'aprendre correctament (**underfit**) les característiques del dataset quan les línies d'entrenament i validació siguin línies rectes. Per contra, si la tendència és la correcta i descendeix, haurem de revisar si existeix la possibilitat que amb més entrenaments segueixi descendent fins a arribar a un punt de saturació.

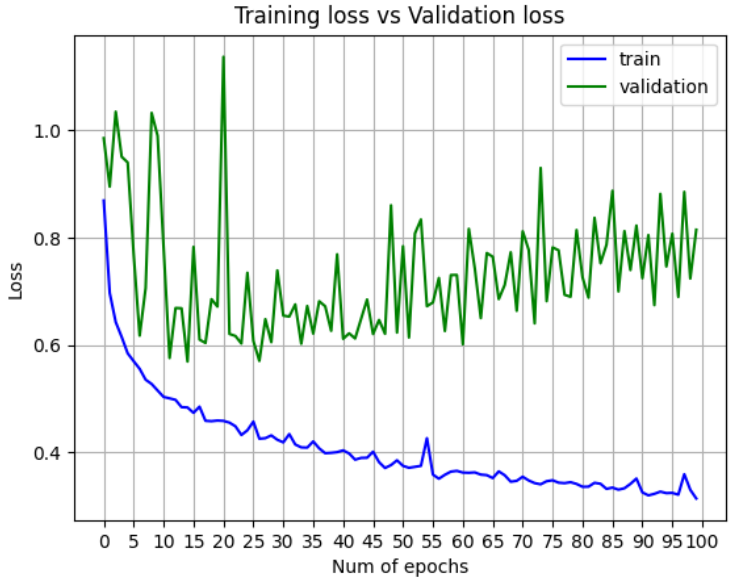



*Il·lustració 27. Exemple de model underfit*




*Il·lustració 28. Exemple de model underfit que necessita més èpoques d'entrenament*

Existeix el cas contrari on un model ha après massa bé el conjunt de dades (**overfit**) d'entrenament, incloent-hi soroll i informació irrellevant, ocasionant una mala generalització del model i fent que no sigui òptim per a qualsevol mena d'imatge. Aquest efecte pot ser causat per la realització d'entrenaments massa llargs o bé per l'ús de models massa sofisticats per prediccions senzilles. Visualment aquest tipus de problema s'anuncia quan la línia de train tendeix a 0 i la de validation s'allunya i té molta distorsió.




*Il·lustració 29. Exemple de model sobre ajustat al conjunt de dades extret d'un entrenament fallit*

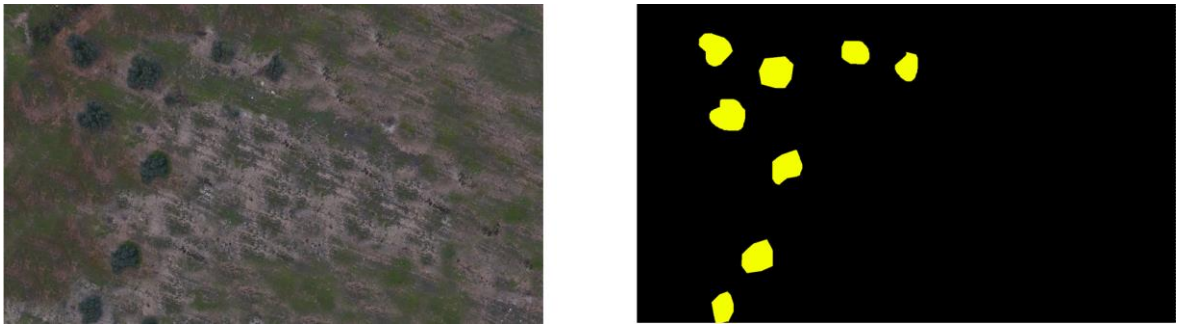
En cas que l'entrenament sigui satisfactori, les línies tindran una tendència descendent, pròxima a 0 i amb resultats semblants.

### *3.3.3.2. Paràmetres de mesura en el testeig*

És molt important elegir unes bones mètriques per poder quantificar l'èxit dels diferents experiments.

En moltes ocasions la mètrica més utilitzada és el **píxel accuracy**, aquest indicarà el percentatge de píxels que s'han predit correctament en comparar les diferents prediccions amb els seus respectius ground truth. Amb aquesta mètrica s'ha d'anar amb compte, ja que en imatges on hi ha molt fons i els objectes a predir són molt petits es podria donar un índex de predicció molt alt quan realment la predicció de l'objecte ha sigut baixa.

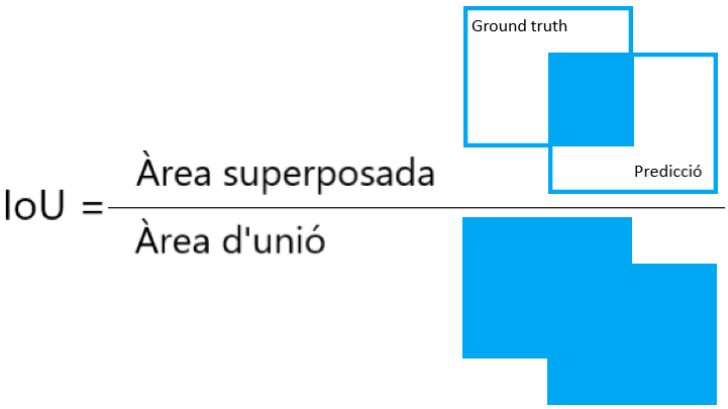
Tal com podem veure a la Il·lustració 30, si per aquesta imatge la seva predicció fos la mostrada en la Il·lustració 31, indicaria un 95% d'èxit perquè la majoria dels píxels (fons d'imatge) han sigut predits correctament, però les mostres d'arbres no.



*Il·lustració 30. A l'esquerra una imatge d'un terreny i a la dreta el seu ground truth*

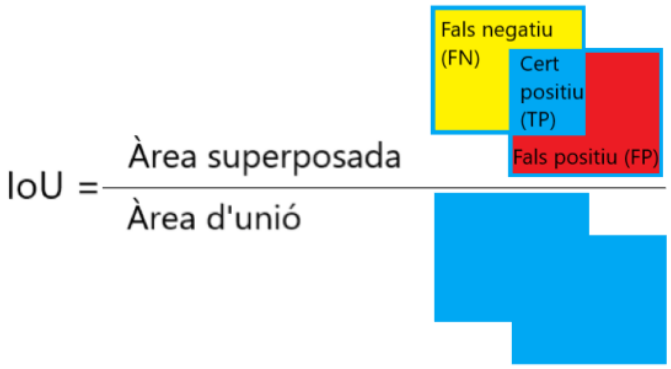
*Il·lustració 31. Predicció de la Il·lustració 30 amb 95% d'èxit.*

Tenint en compte el tipus d'imatge que volem predir i els problemes esmentats anteriorment, en el nostre cas les mètriques presses es basaran en l'**Input over Union (IoU)** o índex Jaccard. Aquest s'utilitza per mesurar la similitud que hi ha entre diferents conjunts de mostres i ve definit per l'àrea de superposició entre la predicció i el ground truth dividida per l'àrea d'unió entre la predicció i el ground truth.




*Il·lustració 32. Exemple d'intersecció sobre la unió*

On a la segmentació semàntica d'objectes, tal com podem veure a la Il·lustració 33, l'àrea de superposició fa referència als píxels etiquetats correctament (TP) i l'àrea d'unió fa referència als píxels segmentats erròniament en la predicció (àrea vermella), píxels no segmentats en la predicció (àrea groga) i els píxels segmentats correctament (àrea blava).




*Il·lustració 33. Exemple d'intersecció sobre la unió per a segmentació semàntica*

I ve definit per la següent equació:

$$IoU = \frac{TP}{TP + FP + FN}$$

*Equació 3. Càlcul de IoU per la segmentació semàntica d'imatges*

D'aquesta manera mesurarem l'èxit de predicció pels diferents objectes: el terra (**Ground IoU**), l'herba (**Grass IoU**) i els arbres (**Tree IoU**).

El **mean IoU** és la mitjana aritmètica del IoU de tots els objectes classificats i ens indicarà com de bona ha sigut la predicció en termes generals.

El **frequency weighted IoU** és el sumatori dels IoU de cada classe tenint en compte el seu pes dins de la imatge, aquesta mesura és la que ens donarà un resultat més acurat de la predicció real.

# 4. Avaluació

Arribats aquest punt és moment de dur a terme l'execució dels experiments i validació dels resultats.

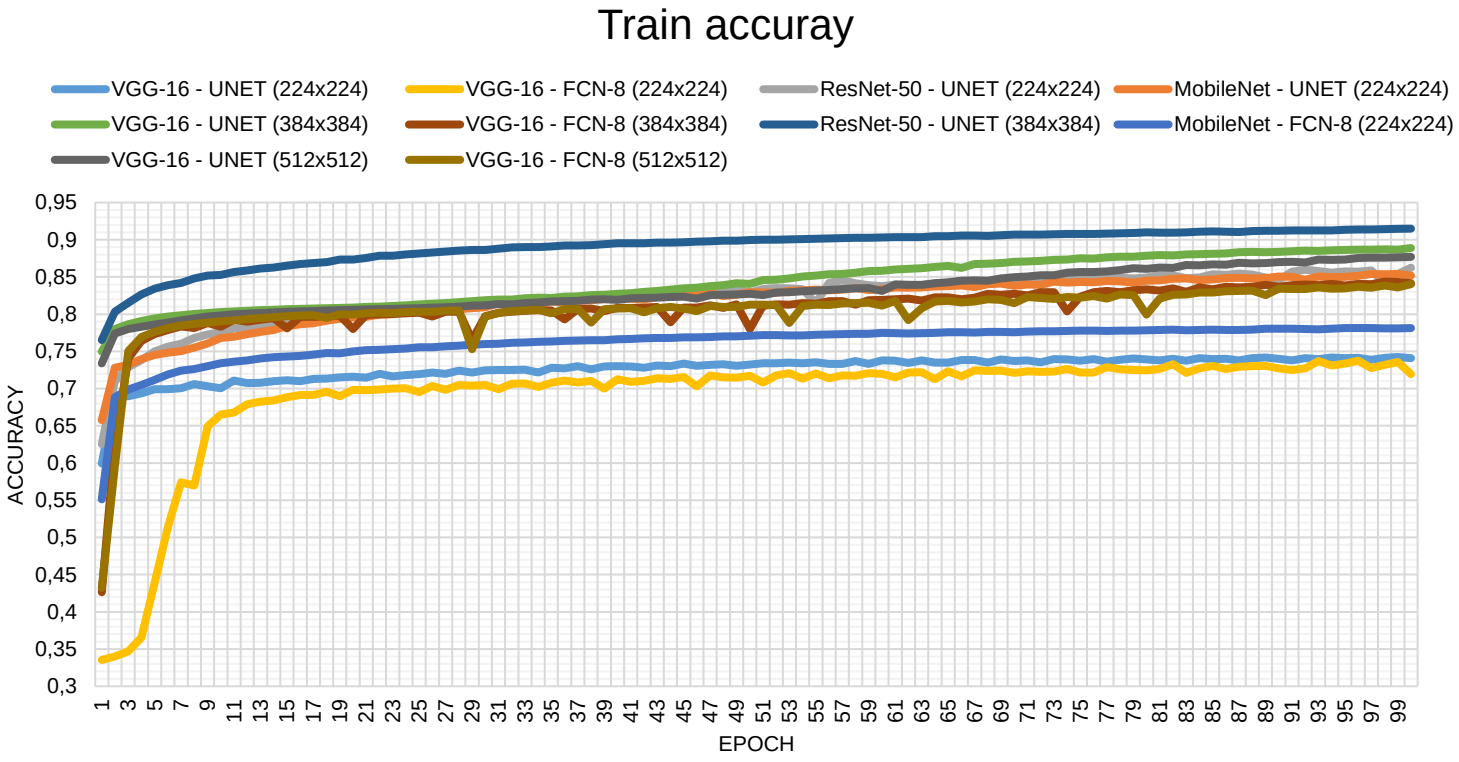
Cal avançar que algunes proves amb ResNet-50 no han pogut ser realitzades satisfactòriament per la falta de recursos; la GPU utilitzada no tenia memòria suficient per entrenar aquest model. També el model MobilNet donada la seva definició únicament es pot entrenar amb imatges de mida 224x224.

## 4.1. Avaluació de l'entrenament

Seguint els passos indicats l'Annex B - Script de l'experiment, carregarem i entrenarem els diferents models i arquitectures amb els conjunts de dades corresponents.

A partir de les dades obtingues en els entrenaments, adjuntes a l'**Annex C - Resultat dels entrenaments**, hem generat la següent  Gràfica 1 on es pot apreciar que el model que presenta **millor precisió d'entrenament** ha estat el **Resnet-50 amb arquitectura UNET** per imatges de 384x384, aquest és un resultat previsible donat que es tracta del model que compta amb més capes per la qual cosa hauria d'aprendre millor que la resta i així mateix l'arquitectura implementada és l'evolució de la FCN-8.

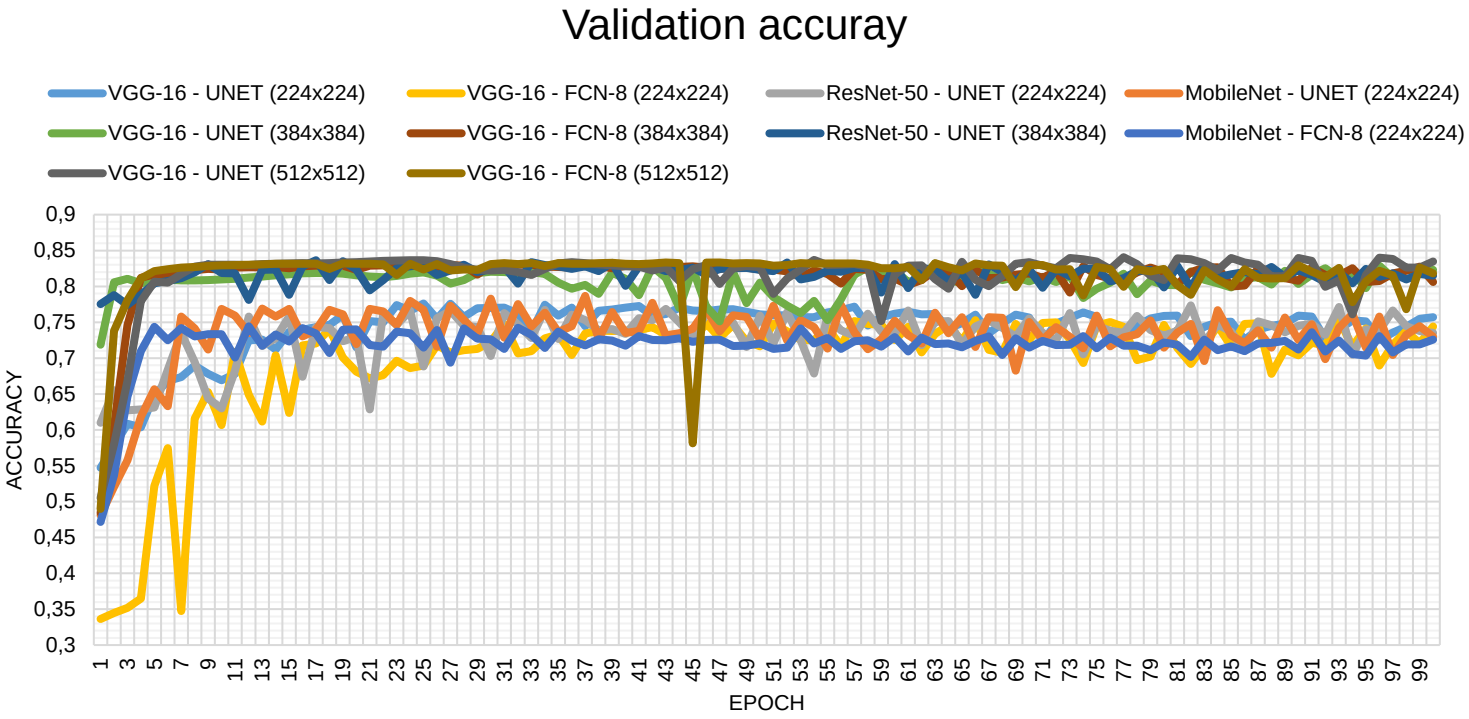
Això únicament ens indica que aquest model ha sigut el que millor ha après a interpretar les imatges durant el procés d'entrenament, però en cap moment indica que és la millor opció, per això serà necessari comparar aquest resultat amb els resultats de validació (Gràfica 2) per veure si generalitza correctament.




*Gràfica 1. Agrupació dels resultats de la precisió d'entrenament pel conjunt de dades train*

En general tots els entrenaments tenen una correcta evolució, però els entrenaments que utilitzen imatges petites (224x224) són els que representen pitjors resultats, tenint 3 dels 5 respectius entrenaments en últimes posicions.

A continuació a la Gràfica 2 s'adjunta els resultats de precisió obtinguts a l'entrenament pel conjunt de dades de validació. Si els resultats amb els mencionats anteriorment apreciem que alguns models presenten problemes d'overfiting i de saturació en l'entrenament; entrarem en detall més endavant.




*Gràfica 2. Agrupació dels resultats de la precisió d'entrenament pel conjunt de dades validation*

A la Taula 2 es mostra un resum dels diferents entrenaments executats destacant el temps d'execució requerit per l'entrenament així com els valors màxims obtinguts en la predicció pel conjunt de dades d'entrenament i de validació.

| Model | Architecture | Input size | Max. Train Pixel Accuracy | Max. Val. Pixel Accuracy | Execution time |
|---|---|---|---|---|---|
| VGG-16 | FCN-8 | 224x224 | 0,7375 | 0,7529 | 3 h 8 m 8.770 s |
| | | 384x384 | 0,7375 | 0,8302 | 8 h 19 m 26.73 s |
| | | 512x512 | 0,8404 | 0,8331 | 12 h 7 m 11.79 s |
| | UNET | 224x224 | 0,7425 | 0,7761 | 1 h 39 m 52.16 s |
| | | 384x384 | 0,8889 | 0,8319 | 5 h 27 m 24.68 s |
| | | 512x512 | 0,877 | **0,8407** | 7 h 53 m 29.54 s |
| ResNet-50 | FCN-8 | 224x224 | Error | Error | Error |
| | | 384x384 | Error | Error | Error |

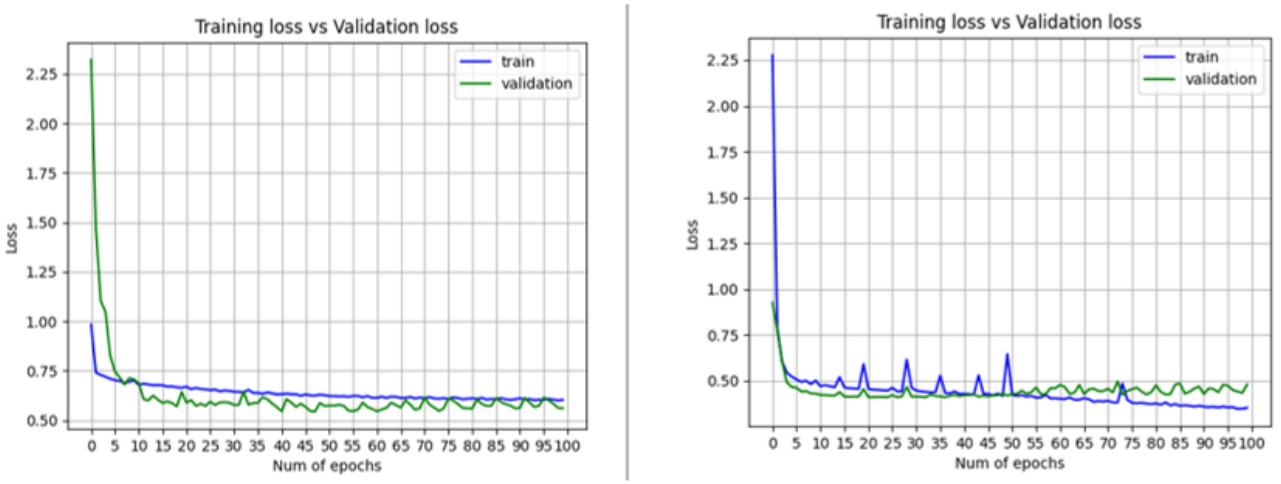
| | | | | | |
|---|---|---|---|---|---|
| | | 512x512 | Error | Error | Error |
| | UNET | 224x224 | 0,8624 | 0,7735 | **1 h 29 m 8.131 s** |
| | | 384x384 | **0,9149** | 0,8364 | 4 h 10 m 28.29 s |
| | | 512x512 | Error | Error | Error |
| MobileNet | FCN-8 | 224x224 | 0,7813 | 0,7442 | 4 h 43 m 28.53 s |
| | | 384x384 | - | - | - |
| | | 512x512 | - | - | - |
| | UNET | 224x224 | 0,854 | 0,7867 | 55 m 11.86 s |
| | | 384x384 | - | - | - |
| | | 512x512 | - | - | - |

*Taula 2. Valors obtinguts en els entrenaments*

Tenint en compte que els entrenaments s'han realitzat per a 100 èpoques i que a cada època s'analitzaven 1400 imatges, el temps requerit per realitzar els entrenaments és positiu, ja que per exemple pels casos on la mida d'imatge és la més petita (224x224) ens surt aproximadament que **1 època de 1400 imatges no triga més de 2 min en executar-se**; alhora aquest valor és fàcilment millorable amb tècniques d'aturada preventives i utilitzant recursos que disposin de més memòria i de més d'un nucli per realitzar les execucions.

Unes altres dades molt importants a analitzar dels resultats adjuntats a **l'Annex C - Resultat dels entrenaments** son les **gràfiques de pèrdua de l'entrenament**.

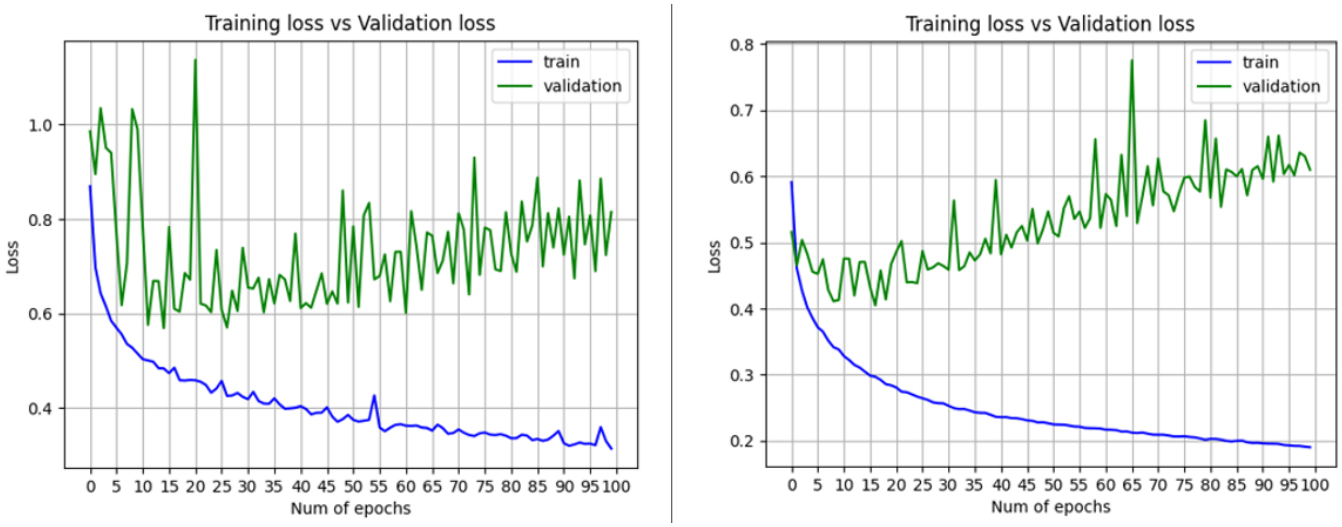
La pèrdua del model **VGG-16 amb FCN-8** per entrades de 224x244 mostra molt bons resultats indicant una **bona generalització**, però s'aprecia a partir de l'època 65 una saturació on el model deixa d'aprendre indicant així que l'entrenament es podria haver finalitzat abans, però tot i això el resultat no empitjora. Així mateix per les entrades 384x384 i 512x512 la generalització també és bona però s'aprecia que si l'entrenament hagués sigut més llarg, el model hauria quedat sobre entrenat oferint una mala generalització i una excessiva divergència entre les mètriques.




*Gràfica 3. Resultats de pèrdues de l'entrenament VGG-16 amb FCN-8. A la esquerra 224x224 i a la dreta 384x384*

La pèrdua del model **VGG-16 amb UNET** per entrades de **224x244** mostra **bons resultats** però l'entrenament podria haver sigut més curt. Per contra, per a entrades **384x384 i 512x512** els **resultats són dolents**, l'excessiva divergència i els constants pics de la gràfica de validació indiquen que el model **no generalitza correctament.**

Tot i que el model **ResNet-50** ha sigut el model que ha obtingut millor precisió d'entrenament, els resultats de pèrdua ens indica que aquest model **no generalitza correctament** i només serà capaç de realitzar bones prediccions a imatges que s'assemblin a les incloses al dataset d'entrenament.




*Gràfica 4. Resultats de pèrdues de l'entrenament ResNet-50 amb UNET. A la esquerra 224x224 i a la dreta 384x384*

**MobileNet** tant per arquitectura FCN-8 i UNET, els resultats no són favorables ja que ens indiquen que **no generalitza correctament**.

Com hem pogut veure, els resultats són molt variants però encara no estem en disposició de fer una elecció ja que per això necessitarem analitzar les dades resultants del testeig contra el dataset de test.

## 4.2. Avaluació del testeig

Un cop entrenat els diferents models, és moment de provar amb el dataset de test i analitzar el seu rendiment amb la predicció d'imatges que no coneix. A continuació a la Taula 3 es mostren els resultats obtinguts a les proves que tenen en compte com de precisa ha estat la predicció del model des de diferents perspectives.

| Model | Architecture | Input size | Frequency weighted IoU | Mean IoU | Ground IoU | Tree IoU | Grass IoU |
|---|---|---|---|---|---|---|---|
| VGG-16 | FCN-8 | 224X224 | 0,5611 | 0,5635 | 0,5989 | 0,4331 | 0,6586 |
| | | 384X384 | 0,6822 | 0,7076 | **0,6990** | 0,8027 | 0,6212 |
| | | 512X512 | **0,6846** | 0,7081 | 0,6934 | 0,7952 | 0,6356 |
| | UNET | 224X224 | 0,5859 | 0,5890 | 0,6222 | 0,4373 | **0,7075** |
| | | 384X384 | 0,6819 | **0,7091** | 0,6967 | 0,8104 | 0,6202 |
| | | 512X512 | 0,6761 | 0,7063 | 0,6761 | **0,8179** | 0,6248 |
| ResNet-50 | FCN-8 | 224X224 | Error | Error | Error | Error | Error |
| | | 384X384 | Error | Error | Error | Error | Error |
| | | 512X512 | Error | Error | Error | Error | Error |
| | UNET | 224X224 | 0,5667 | 0,5687 | 0,6173 | 0,4466 | 0,6423 |
| | | 384X384 | 0,6724 | 0,6949 | 0,6881 | 0,7789 | 0,6177 |
| | | 512X512 | Error | Error | Error | Error | Error |
| MobileNet | FCN-8 | 224X224 | 0,5624 | 0,5649 | 0,6057 | 0,4264 | 0,6626 |
| | | 384X384 | - | - | - | - | - |
| | | 512X512 | - | - | - | - | - |
| | UNET | 224X224 | 0,5537 | 0,5560 | 0,6078 | 0,4172 | 0,6431 |
| | | 384X384 | - | - | - | - | - |
| | | 512X512 | - | - | - | - | - |

*Taula 3. Resultat dels experiments*

De tots els models entrenats, l'opció **VGG-16 amb UNET** per a imatges de mida **512x512** és el que realitza la millor **predicció d'arbres** oferint un èxit del **81,79%**.

Si ens fixem en termes més genèrics, l'opció que realitza una **millor predicció de tots els objectes** amb un èxit del **70,91%** és el model **VGG-16 amb arquitectura UNET** per a imatges de mida **384x384**.

Si tenim en compte el tipus d'imatge que estem tractant i la freqüència amb què apareixen aquests objectes en escena, la millor opció amb un èxit del **68,46%** fa referència al model **VGG-16 amb arquitectura FCN-8** per a imatges de mida **512x512**.

Per altra banda, és curiós que **ResNet-50** tot i haver estat els dos experiments amb pitjor resultat de pèrdua i anunciar una poca generalització del model, han donat bons resultats si els comparem amb models que no han tingut aquest tipus de problema.

Tenint en compte que el model predictiu elegit haurà d'estar en un entorn productiu, serà important analitzar i validar el seu rendiment per així conèixer les limitacions.

| Model | Architecture | Input size | Model size | Execution time |
|---|---|---|---|---|
| VGG-16 | FCN-8 | 224X224 | 1,5 GB | 0,1753 s/it |
| | | 384X384 | 1,5 GB | 0,2363 s/it |
| | | 512X512 | 1,5 GB | 0,3357 s/it |
| | UNET | 224X224 | 141 MB | 0,0786 s/it |
| | | 384X384 | 141 MB | 0,1571 s/it |
| | | 512X512 | 141 MB | 0,2071 s/it |
| ResNet-50 | FCN-8 | 224X224 | Error | Error |
| | | 384X384 | Error | Error |
| | | 512X512 | Error | Error |
| | UNET | 224X224 | 187 MB | 0,0857 s/it |
| | | 384X384 | 187 MB | 0,1357 s/it |
| | | 512X512 | Error | Error |
| MobileNet | FCN-8 | 224X224 | 95,7 MB | 0,1022 s/it |
| | | 384X384 | - | - |
| | | 512X512 | - | - |
| | UNET | 224X224 | **72,2 MB** | **0,0571 s/it** |
| | | 384X384 | - | - |
| | | 512X512 | - | - |

*Taula 4. Resultat de les mides dels models i el temps de predicció per mostra*

Observant la Taula 4 podem veure el temps que triga a processar una mostra predicció el model predictiu, per defecte els models que processaran una imatge més petita, seran més ràpids.

Tenint en compte el nostre cas, tal com hem gestionat el dataset 35 imatges formen una captura de dron, sent així que en el millor dels casos sense tenir en compte demores per retallar i redimensionar la imatge, **predir una captura de dron** ens demoraria aproximadament **2 segons**, que en aquest cas concret coincideix amb la freqüència de captura de les imatges. Per tant, pensem que és viable aplicar el model en aplicacions de temps real. A més, com que hi ha un solapament significatiu entre imatges consecutives (un 80%), també podríem optar per no processar totes les imatges.

Finalment, un cop analitzats tots els resultats, resulta difícil decidir quin és el millor model, ja que dependrà de l'aplicació d'ús final i dels requisits del sistema els que marcaran quin és el millor model a elegir.

Tot i això, considerem que **la millor opció per un cas genèric** seria el **model ResNet-50 amb arquitectura UNET per imatges de 384x384** perquè treballem amb imatges no excessivament grans, el model té un temps d'entrenament moderat que facilitarà els entrenaments i tenint en compte les gràfiques de funció de pèrdua de l'entrenament, creiem que hi ha marge de millora. També el pes del model és moderat, 187 MB i el temps de processament d'una imatge és de 4,75 segons.

Tot i així, si observéssim que aquest model no acaba de generalitzar correctament, l'opció del **model VGG-16 amb arquitectura UNET per imatges de 384x384** és una opció molt interesant i amb els resultats casi idèntics que el ResNet-50.

# Conclusions

En el present treball hem estudiat les bases de l'anàlisi d'imatges fent ús de tècniques de deep learning enfocades a segmentació semàntica d'imatges, aprenent a fer ús dels diferents models i arquitectures que conformen una xarxa neuronal convolucional.

Dels resultats podem extreure que per a entorns d'anàlisis d'imatge en temps real amb una baixa potència de software, el model **MobileNet** és el més adequat, tot i tindre l'inconvenient del seu rati d'aprenentatge.

A mesura que els requisits siguin més flexibles, podrem ampliar l'elecció a models amb millors resultats com ResNet i VGG-16.

Tenint en compte les dificultats a l'hora de gestionar les dades, una bona opció seria l'ús del model **ResNet-50 amb arquitectura UNET per mides d'imatge de 384x384** amb el que podríem processar imatges en diferit durant el vol o bé a les parades tècniques fent ús d'un dispositiu extern.

Basant-nos en l'experiència i els resultats obtinguts, en futurs projectes descartaríem l'ús de models densos amb arquitectures FCN-8 a causa dels alts recursos que exigeix; en aquests casos es recomanaria l'ús de **l'arquitectura UNET**, ja que el fet d'implementar capes d'omissió **redueix els requisits significativament**.

Respecte a la mida de les imatges dels diferents conjunts de dades, els resultats han demostrat que com més gran és la mida de la imatge, es requerirà més temps i memòria per entrenar, d'igual forma el temps d'execució en un entorn productiu serà més elevat. En el nostre cas hem considerat òptim el terme mitja (384x384), però creiem que serà necessari en cada cas avaluar quin s'adequa millor.

També hem vist la importància de l'augment de dades i com per una tasca senzilla amb només 10 imatges hem aconseguit obtindré uns resultats acceptables per poder iniciar una primera fase d'implementació.

Tot i això, molts dels nostres models indicaven indicis d'overfiting fruit de la mancança de dades, ressaltant la importància de què és necessari generar un conjunt de dades divers per obtenir bons resultats davant casos més complexos.

Finalment, en cas de voler implementar l'anàlisi d'imatge en temps real en un dispositiu un UAV de baix cost, es recomana continuar l'estudi amb la recerca d'arquitectures i models més lleugers que ofereixin un temps de predicció inferior als obtinguts, com també investigar tècniques de transferència de pesos per utilitzar models pre-entrenats o inclús combinar entrenaments amb un conjunt de dades de mida variable, donat que alguns models demostren millor resultats davant aquest tipus d'entrenament [10].

# Bibliografia

[1] John McCarthy, “WHAT IS ARTIFICIAL INTELLIGENCE?,” *November 12, 2007*, 2007. http://www-formal.stanford.edu/jmc/whatisai/node1.html (accessed Oct. 25, 2019).

[2] E. Salamí, A. Gallardo, G. Skorobogatov, and C. Barrado, “On-the-fly olive tree counting using a UAS and cloud services,” *Remote Sens.*, vol. 11, no. 3, p. 316, Feb. 2019, doi: 10.3390/rs11030316.

[3] F.-F. Li, A. Karpathy, and J. Johnson, “Spatial Localization and Detection,” *Stanford Convolutional Neural Networks Vis. Recognit.*, pp. 1–90, Feb. 2016, Accessed: Mar. 11, 2020. [Online]. Available: http://cs231n.stanford.edu/slides/2016/winter1516_lecture8.pdf.

[4] A. Arnab *et al.*, “Conditional Random Fields Meet Deep Neural Networks for Semantic Segmentation: Combining Probabilistic Graphical Models with Deep Learning for Structured Prediction,” *IEEE Signal Process. Mag.*, vol. 35, no. 1, pp. 37–52, 2018, doi: 10.1109/MSP.2017.2762355.

[5] “Redes Neuronales Convolucionales - MATLAB & Simulink,” *https://la.mathworks.com/*, 2019. https://es.mathworks.com/solutions/deep-learning/convolutional-neural-network.html (accessed Jun. 19, 2020).

[6] F. Rosner, “Handwritten Digit Recognition Using Convolutional Neural Networks,” *May 28, 2018*, May 28, 2018. https://dev.to/frosnerd/handwritten-digit-recognition-using-convolutional-neural-networks-11g0 (accessed Mar. 10, 2020).

[7] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-Decem, pp. 770–778, 2016, doi: 10.1109/CVPR.2016.90.

[8] A. Berg, J. Deng, and L. Fei-Fei, “ImageNet Large Scale Visual Recognition Competition (ILSVRC),” 2010. http://www.image-net.org/challenges/LSVRC/ (accessed Jun. 18, 2020).

[9] Jesus Utrera Burgal, “Deep Learning básico con Keras (Parte 4): ResNet,” 2018. https://enmilocalfunciona.io/deep-learning-basico-con-keras-parte-4-resnet/ (accessed Jul. 02, 2020).

[10] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” 2015, Accessed: Jun. 18, 2020. [Online]. Available: http://www.robots.ox.ac.uk/.

[11] A. G. Howard *et al.*, “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications.”

[12] J. Long, E. Shelhamer, and T. Darrell, “Fully Convolutional Networks for Semantic Segmentation.”

[13] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical

image segmentation," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2015, vol. 9351, pp. 234–241, doi: 10.1007/978-3-319-24574-4_28.

[14] V. Badrinarayanan, A. Kendall, and R. Cipolla, "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 12, pp. 2481–2495, 2017, doi: 10.1109/TPAMI.2016.2644615.

[15] F. Chollet, "Keras: the Python deep learning API," 2017. https://keras.io/ (accessed Jun. 07, 2020).

[16] T. Xu *et al.*, *TensorFlow: A System for Large-Scale Machine Learning*. 2016.

[17] D. Gupta, "GitHub - segmentation-keras." Divam Gupta, Accessed: Apr. 01, 2020. [Online]. Available: https://github.com/divamgupta/image-segmentation-keras/.

[18] B. Russell, A. Torralba, and W. T. Freeman, "Labelme: The open annotation tool," *Computer Science and Artificial Intelligence Laboratory [Online]. Available: http://labelme. csail. mit. edu*, 2006. http://labelme.csail.mit.edu/Release3.0/ (accessed Feb. 20, 2020).

[19] Amazon, "Amazon SageMaker Ground Truth | AWS." https://aws.amazon.com/es/sagemaker/groundtruth/ (accessed Mar. 12, 2020).

[20] "GIMP - GNU Image Manipulation Program." https://www.gimp.org/ (accessed Mar. 10, 2020).

[21] "Carzyx/Azure-Machine-Learning." https://github.com/Carzyx/Azure-Machine-Learning (accessed Jul. 06, 2020).

[22] F. Chollet, "Building powerful image classification models using very little data," *The Keras Blog*, 2016. https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html (accessed Mar. 10, 2020).

[23] S. C. Wong, A. Gatt, V. Stamatescu, and M. D. McDonnell, "Understanding Data Augmentation for Classification: When to Warp?," 2016, doi: 10.1109/DICTA.2016.7797091.

[24] Y. Wu *et al.*, "A Comparative Measurement Study of Deep Learning as a Service Framework," *IEEE Trans. Serv. Comput.*, p. 15, doi: 10.1109/TSC.2019.2928551.

[25] Google, "Welcome To Colaboratory - Google Colaboratory." https://colab.research.google.com (accessed Mar. 23, 2020).

[26] Google, "Cloud Tensor Processing Unit (TPU)," *Google cloud*. https://cloud.google.com/tpu/docs/tpus (accessed Jun. 06, 2020).

[27] "Symfony - Wikipedia, la enciclopedia libre." https://es.wikipedia.org/wiki/CUDA (accessed Jun. 27, 2020).

[28] M. D. Zeiler, "ADADELTA: An Adaptive Learning Rate Method," 2012, Accessed: Jun. 27, 2020. [Online]. Available: http://arxiv.org/abs/1212.5701.

[29] Microsoft, “Creación de áreas de trabajo de Azure Machine Learning en el portal - Azure Machine Learning | Microsoft Docs.” https://docs.microsoft.com/es-es/azure/machine-learning/how-to-manage-workspace (accessed Apr. 15, 2020).

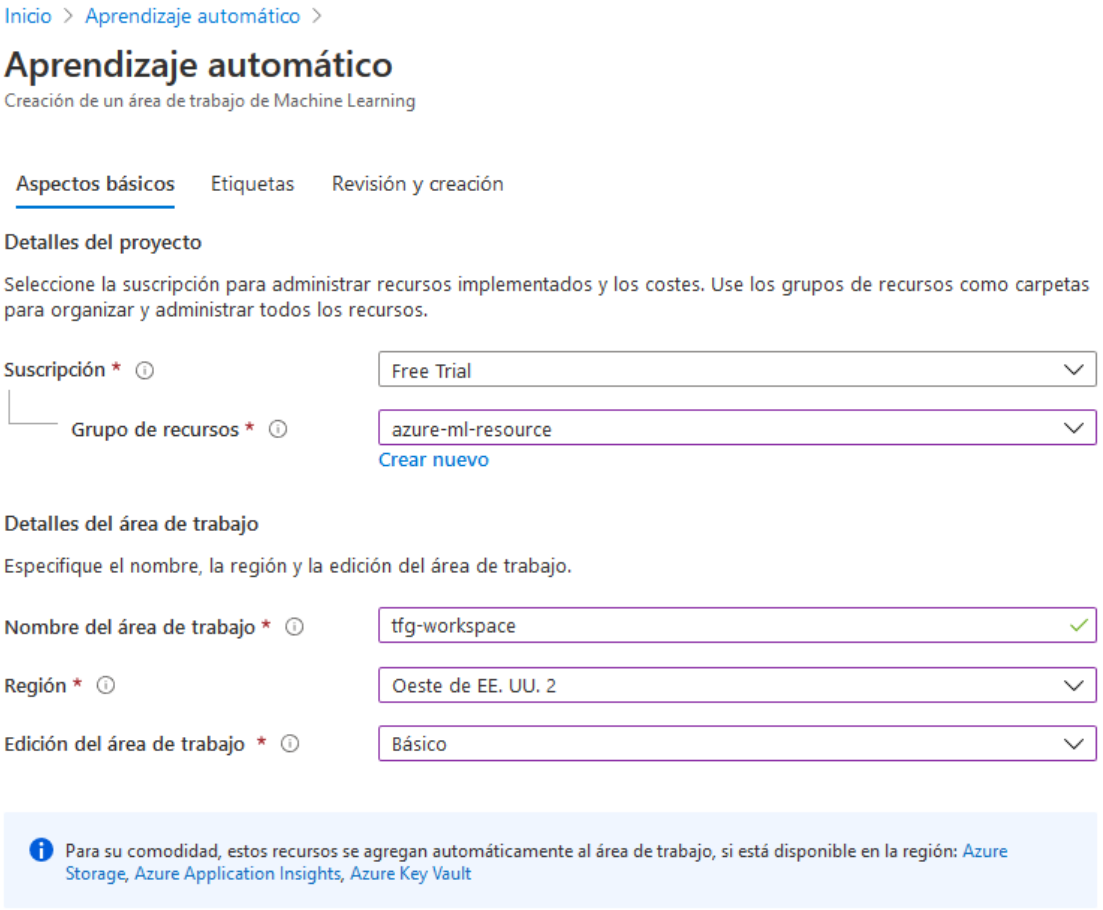
[30] “Azure Products by Region | Microsoft Azure.” https://azure.microsoft.com/en-us/global-infrastructure/services/?products=machine-learning-service,virtual-machines&regions=non-regional,us-east,us-east-2,us-central,us-north-central,us-south-central,us-west-central,us-west,us-west-2 (accessed Jun. 12, 2020).

[31] “Books with Jupyter.” https://jupyterbook.org/intro.html (accessed Jun. 08, 2020).

# Annex A - Configuració d'un entorn Azure Machine Learning

## A1. Creació d'un entorn de treball

Seguint les passes indicades en la documentació oficial de Microsoft [29] crearem una àrea de treball de Azure ML.



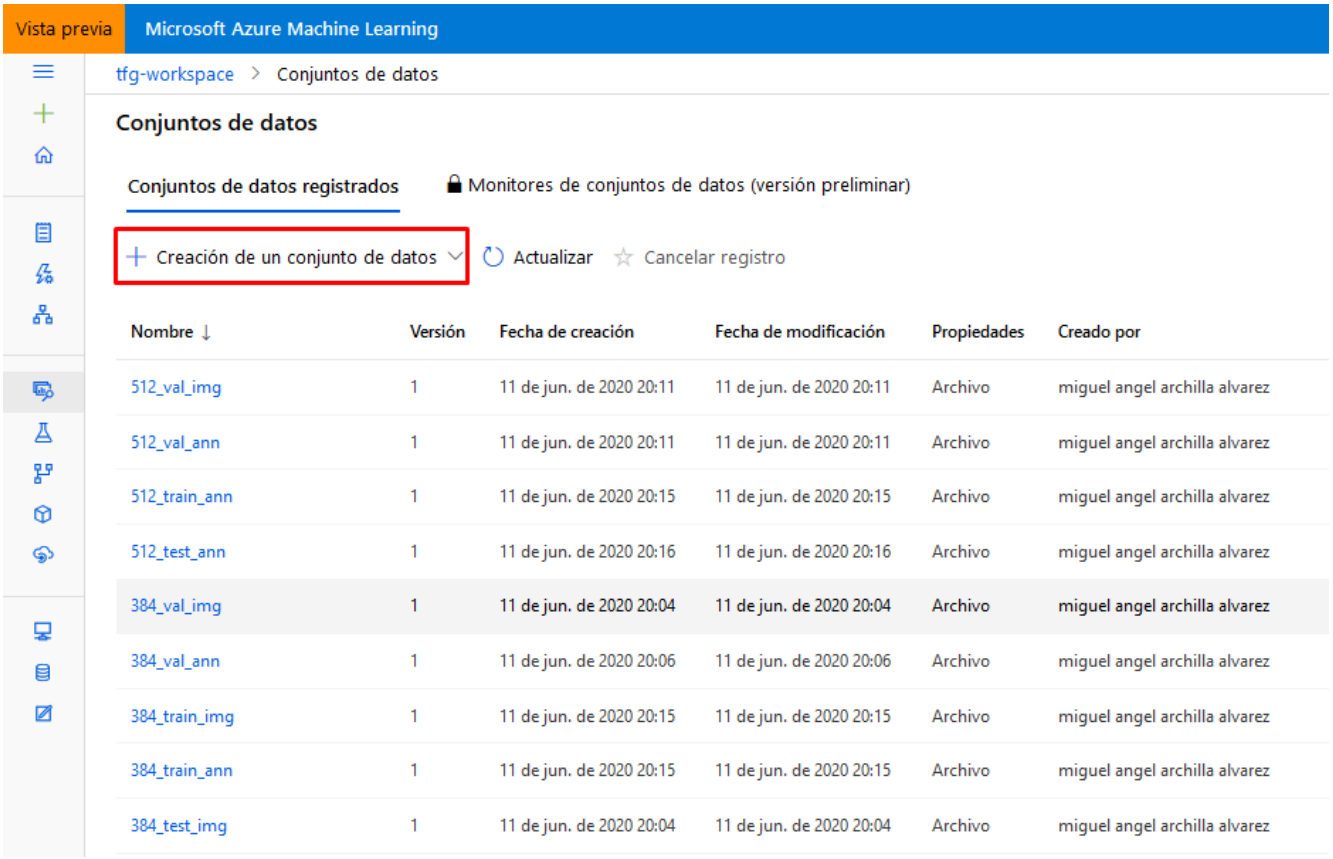

*Il·lustració 34. Creació d'un entorn de Azure Machine Learning*

I a traves del següent enllaç https://ml.azure.com podrem accedir a la nova versió (actualment en fase preliminar) del recursos de Azure ML.

És recomanable revisar abans quina regió volem fer servir [30] donat que els recursos disponibles poden canviar en funció d'aquesta.

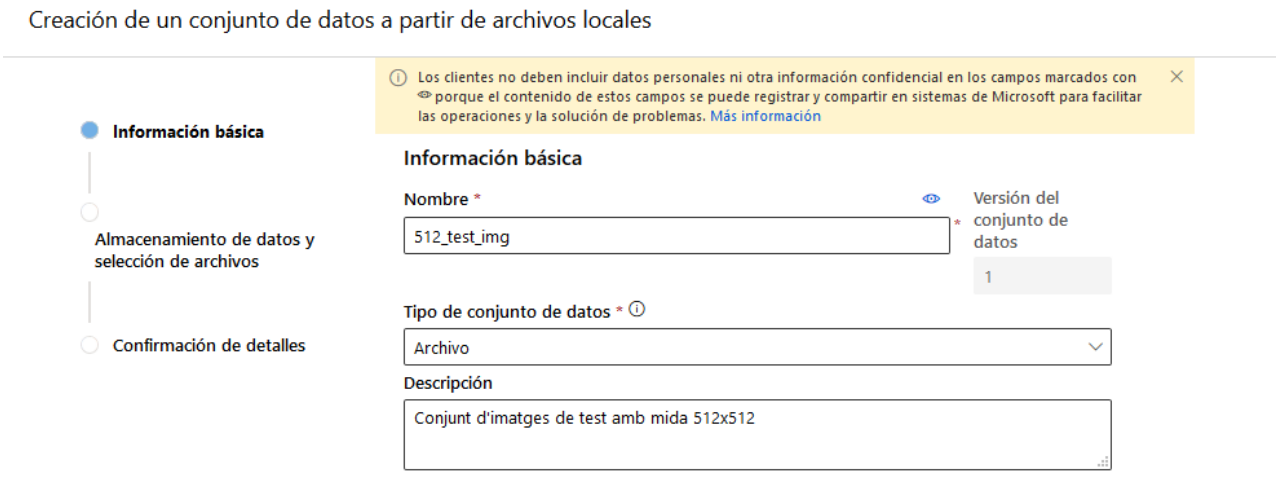
## A2. Creació d'un conjunt de dades

Un cop creada l'àrea de treball, accedirem a ell i crearem els diferents conjunts de dades que necessitarem per executar entrenar i validar els models.

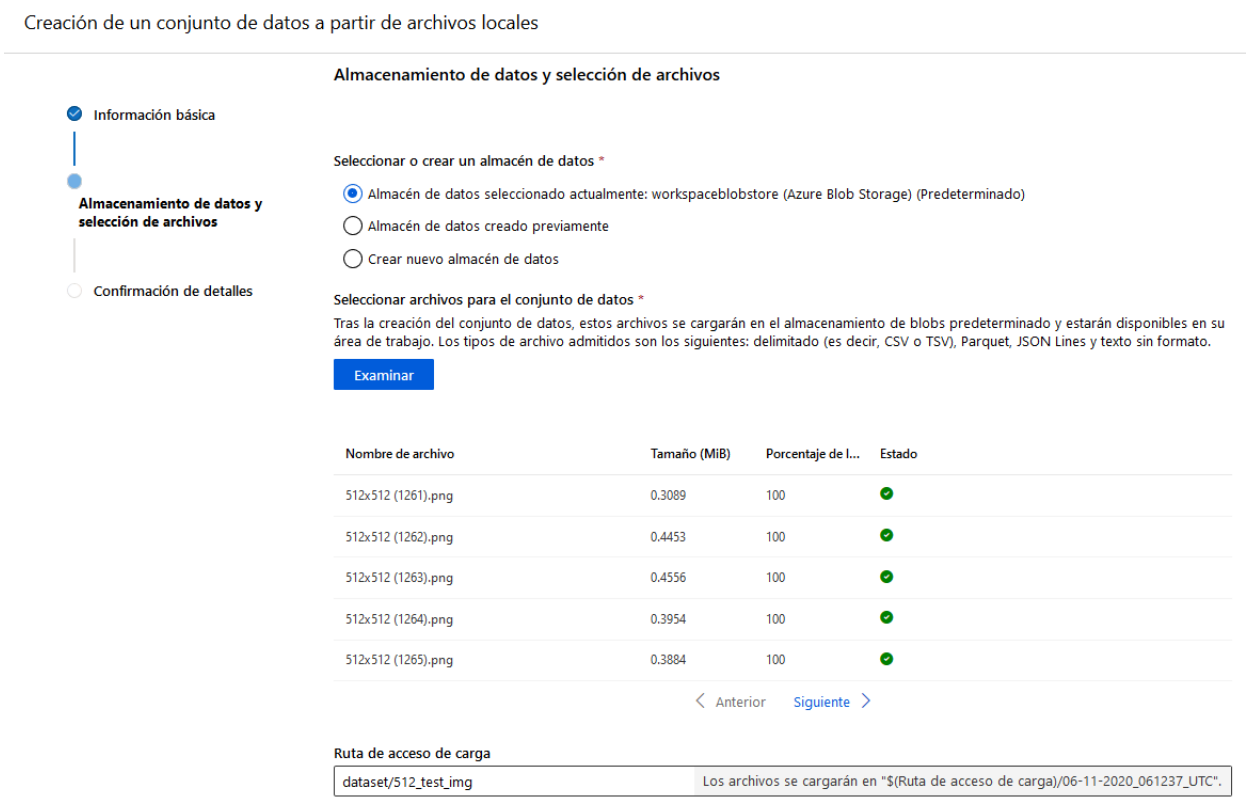



*Il·lustració 35. Pestanya de conjunt de dades*

Seleccionant l'opció *"Creación d'un conjunto de datos"* podrem crear un nou conjunt de dades. Seguidament indicarem com a tipus de dades l'opció de fitxer donat que el nostre dataset està format per imatges, seleccionarem com a recurs d'emmagatzemament l'associat a la nostra àrea de treball, workspaceblobstore de tipus **Azure Blob Storage**, carregarem les imatges manualment del nostre entorn local i indicarem el directori on volem emmagatzemar-les.




*Il·lustració 36. Creació d'un nou conjunt de dades*




*Il·lustració 37. Carrega local del conjunt de dades*

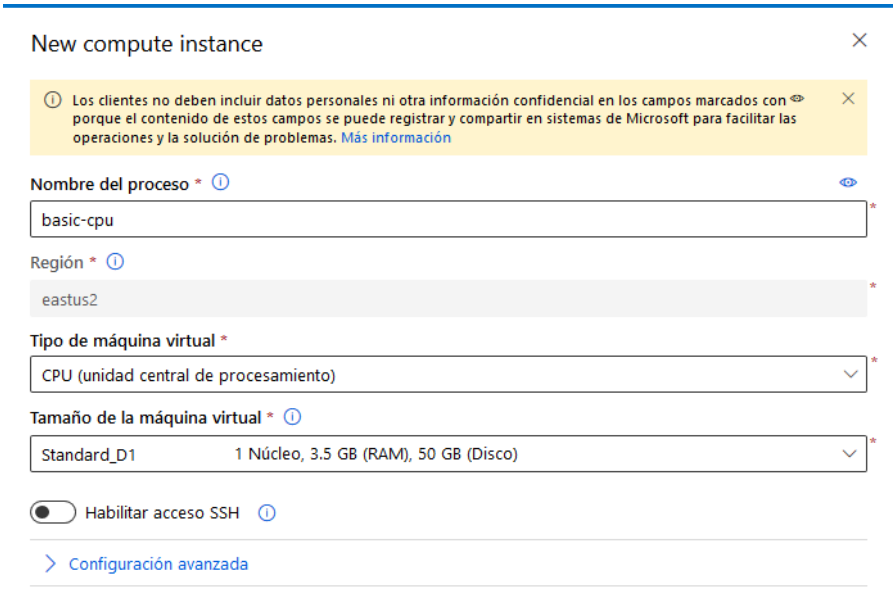
## A3. Creació d'instàncies per l'execució dels experiments

Per dur a terme l'execució de l'experiment necessitarem dues coses:

- Una instància de màquina per poder **preparar i sol·licitar un experiment**.
- Una instància de màquina per **dur a terme l'experiment**.

Dins la pestanya *"Proceso"* en l'apartat *"Instancias de proceso"* definirem una instància de màquina, aquesta serà una CPU de categoria Standard_D1, la més simple de totes, ja que només la necessitem per poder encolar els nostres experiments.

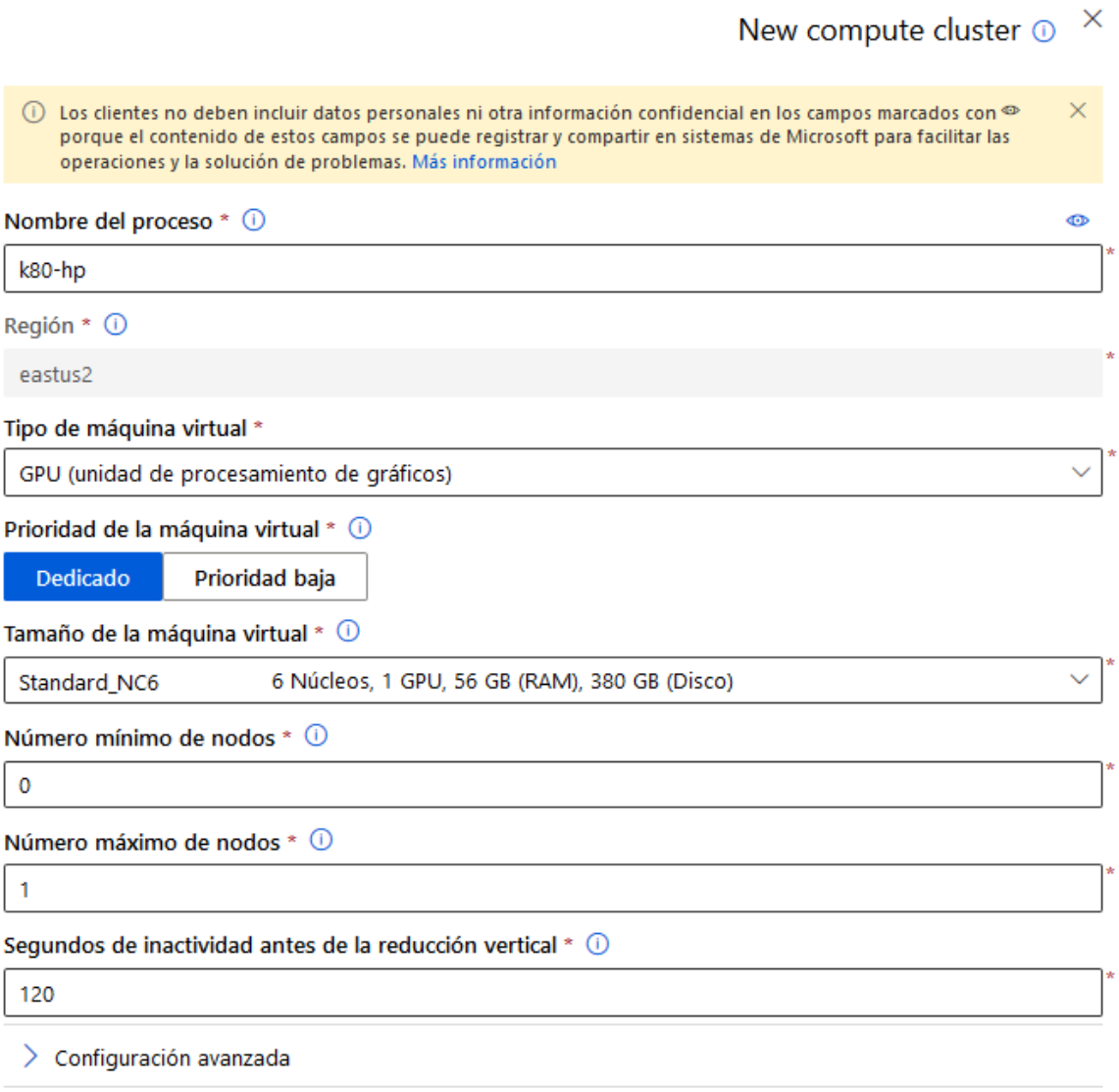
Donat que és una màquina virtual serà important recordar aturar-la un cop no estem fent ús d'ella per tal d'evitar gastos.




*Il·lustració 38. Configuració d'instància de màquina CPU*

Dins la mateixa pestanya, en l'apartat "Clústeres de proceso" definirem les instàncies de màquines que s'encarregaran d'executar els experiments, donada la càrrega computacional aquestes seran màquines GPUs optimitzades per a processos de ML.

Donades les **limitacions d'ús** per part de Microsoft, actualment com a recurs dedicat només podem configurar recursos **Standard_NC6** que contenen GPUs **NVIDIA TESLA K80** i com a màxim podrem disposar d'un node, és a dir, no podrem escalar l'arquitectura per realitzar l'entrenament més ràpid.

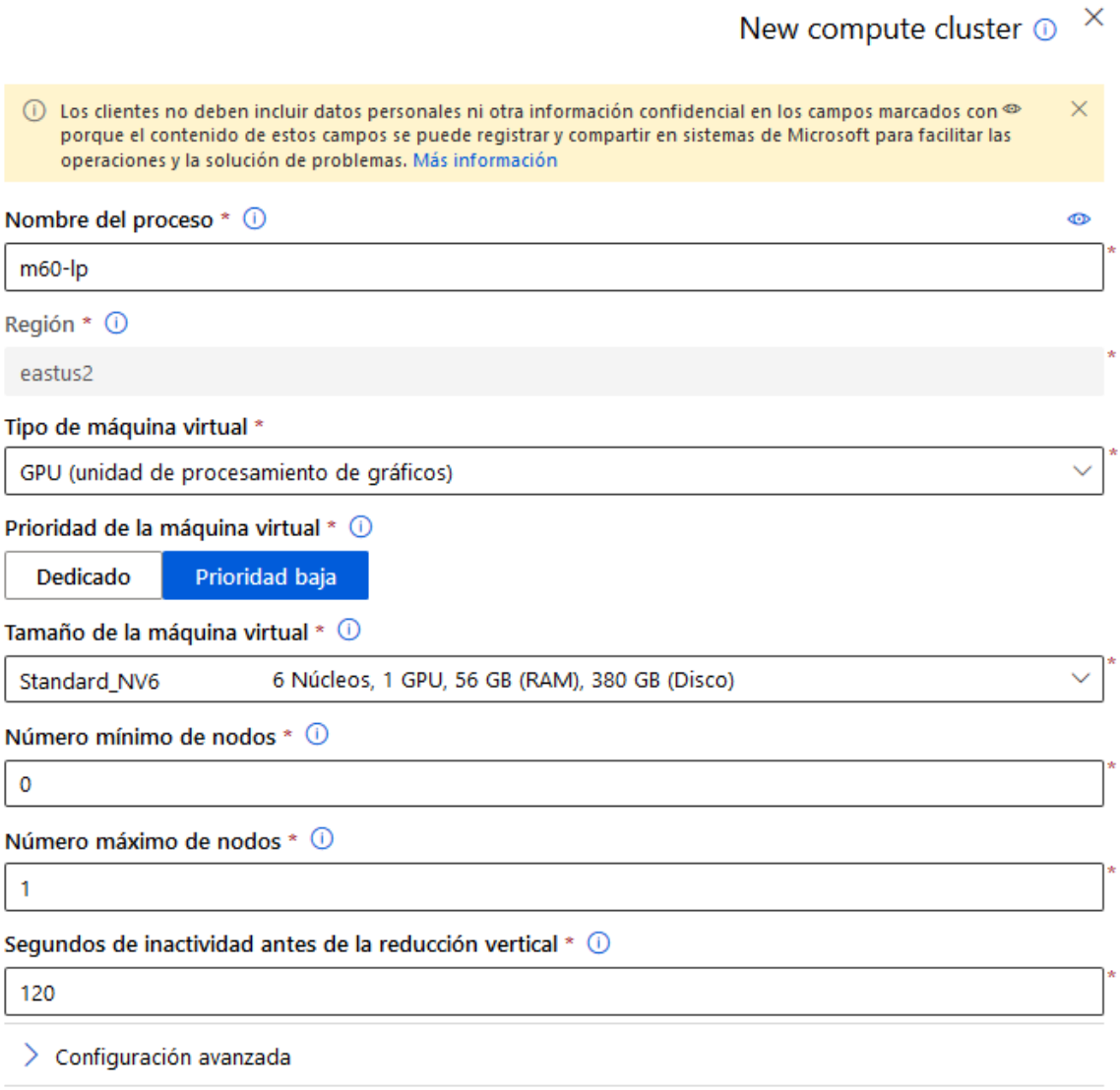



*Il·lustració 39. Configuració d'un recurs GPU dedicat*

Existeix la possibilitat de configurar instàncies de **baixa prioritat** on tindrem disponibles recursos **Standard_NV6** que utilitzen una **NVIDIA Tesla M60**, aquestes instàncies es caracteritzen per tindre un **SLA mínim** on no es garanteix l'ús ininterromput del servei, per la qual cosa no es recomana utilitzar-se en execucions llargues. Tot i això després d'un parell d'execucions vam veure que funcionaven igual de bé i no vam notar cap mena d'interrupció, per aquest motiu recomanem el seu ús i així d'aquesta manera **podrem tindre dos experiments executant-se en paral·lel**.

Preferiblement intentarem sempre executar els entrenaments amb la NVIDIA TESLA K80 donat que és més ràpida i té més memòria GPU que la NVIDIA TESLA M60: 12 GB versus 8 GB.

Tot i això serà convenient revisar si existeixen altres recursos disponibles tenint en compte la regió, ja que Nvidia disposa en el mercat de recursos més potents per dur a terme aquestes tasques.

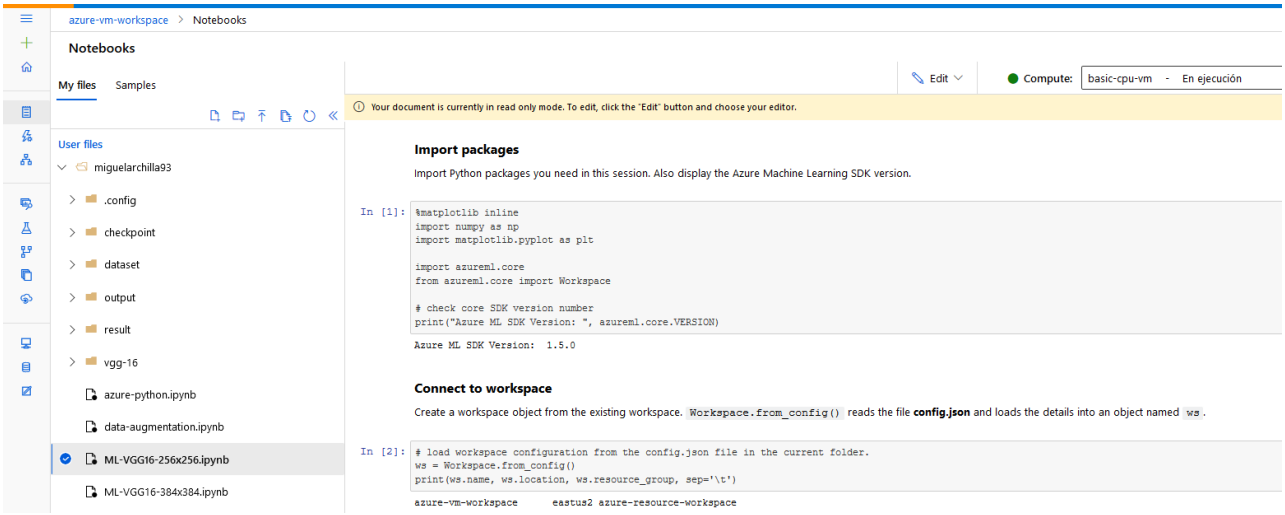



*Il·lustració 40. Configuració d'un recurs GPU de prioritat baixa*

## A4. Creació dels nostres experiments

A la pestanya *"Notebooks"* carregarem el fitxer escrit sobre **Jupyter NoteBook** [31] que contindrà tot el necessari per definir i executar un experiment.

Seguint els diferents tutorials disponibles en la documentació oficial de Microsoft [21] aprendrem a configurar i executar un experiment fent ús de les instàncies prèviament creades per tal d'entrenar un model amb el nostre dataset allotjat a Azure.




*Il·lustració 41. Exemple de experiment carregat*

# Annex B - Script de l'experiment

A continuació s'adjunta el scrpit de l'experiment [21] on s'explica com dur a terme l'entrenament i validació d'un model i arquitectura sobre un entorno Azure ML, consumint un dataset allotjat a Azure Blob Storage i utilitzant els recursos GPUs prèviament definits.

També es mostra la possibilitat d'automatitzar i refinar diversos entrenaments fent ús dels hiperparàmetres.

## B1. Set configuration

### B.1.1 Train configuration

Define primary values to train and test our model.

```
# --- Define values to execute a SINGLE EXPERIMENT ---
single_experiment = True
architecture_name = 'FCN-8'
#Accepted values: FCN-8, UNET
model_name= 'VGG-16'
#Accepted values: VGG-16, Resnet-50, MobileNet
#-------------------------------------------------------------

n_classes = 3
image_size = 384
# Accepted values: 224, 256, 384, 512

if(image_size == 256 or image_size == 224):
    batch_size = 80
    steps_per_epoch = 14
    val_batch_size = 70
    val_steps_per_epoch = 2
    epochs = 100
    print('Loaded 224x224 configuration')
elif(image_size == 384):
    batch_size = 40
    steps_per_epoch = 28
    val_batch_size = 70
    val_steps_per_epoch = 2
    epochs = 100
    print('Loaded 384x384 configuration')
elif(image_size == 512):
    batch_size = 28
    val_batch_size = 40
    steps_per_epoch = 70
    val_steps_per_epoch = 2
    epochs = 100
    print('Loaded 512x512 configuration')
else:
    batch_size = 10
    val_batch_size = 10
    steps_per_epoch = 5
```

```
    val_steps_per_epoch = 5
    epochs = 25
    print('Loaded default configuration to quick test')
```

### B.1.2 Connection & execution params

Define connection and executions params.

```
experiment_folder_name = "TFG"
# Accepted values: 'VGG-16', 'Resnet-50', 'MobileNet'

# Data input prefix associated a the data input name defined
# Our data name is defined by "size_step_type_" for example "384_train_img" or "
384_train_ann"
storage_iput_prefix = str(image_size)

# experiment_name - example: 'VGG-16-384x384'
experiment_name = experiment_folder_name+'-' + \
    storage_iput_prefix+'x'+storage_iput_prefix

# Execution instance name
gpu_instance = "k80-lp"

# Storage credentials
subscription_id = '00000000-0000-0000-0000-00000test'
resource_group = 'azure-ml-resource'
workspace_name = 'tfg-workspace'
```

## B2. Create data & environment

### B.2.1 Import packages

Import Python packages you need in this session. Also display the Azure Machine Learning SDK version.

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

import azureml.core
from azureml.core import Workspace

# check core SDK version number
print("Azure ML SDK Version: ", azureml.core.VERSION)
```

### B.2.2 Connect to workspace

Create a workspace object from the existing workspace. *Workspace.from_config()* reads the file **config.json** and loads the details into an object named *ws*.

```
# load workspace configuration from the config.json file in the current folder.
ws = Workspace.from_config()
print(ws.name, ws.location, ws.resource_group, sep='\t')
```

### B.2.3 Create experiment

Create an experiment to track the runs in your workspace. A workspace can have muliple experiments.

```
from azureml.core import Experiment
exp = Experiment(workspace=ws, name=experiment_name)
```

### B.2.4 Create or Attach existing compute resource

By using Azure Machine Learning Compute, a managed service, data scientists can train machine learning models on clusters of Azure virtual machines. Examples include VMs with GPU support. In this tutorial, you create Azure Machine Learning Compute as your training environment. You will submit Python code to run on this VM later in the tutorial. The code below creates the compute clusters for you if they don't already exist in your workspace.

**Creation of compute takes approximately 5 minutes.** If the AmlCompute with that name is already in your workspace the code will skip the creation process.

```
from azureml.core.compute import AmlCompute
from azureml.core.compute import ComputeTarget
import os

# set a name for your cluster
compute_name = os.environ.get("AML_COMPUTE_CLUSTER_NAME", gpu_instance)
compute_min_nodes = os.environ.get("AML_COMPUTE_CLUSTER_MIN_NODES", 0)
compute_max_nodes = os.environ.get("AML_COMPUTE_CLUSTER_MAX_NODES", 1)

# This example uses CPU VM. For using GPU VM, set SKU to STANDARD_NC6
vm_size = os.environ.get("AML_COMPUTE_CLUSTER_SKU", "STANDARD_NC6")

if compute_name in ws.compute_targets:
    compute_target = ws.compute_targets[compute_name]
    if compute_target and type(compute_target) is AmlCompute:
        print("found compute target: " + compute_name)
else:
    print("creating new compute target...")
    provisioning_config = AmlCompute.provisioning_configuration(vm_size=vm_size,
                                                                 min_nodes=comput
e_min_nodes,
                                                                 max_nodes=comput
e_max_nodes)

    # create the cluster
    compute_target = ComputeTarget.create(
```

```
        ws, compute_name, provisioning_config)

    # can poll for a minimum number of nodes and for a specific timeout.
    # if no min node count is provided it will use the scale settings for the cl
uster
    compute_target.wait_for_completion(
        show_output=True, min_node_count=None, timeout_in_minutes=20)

    # For a more detailed view of current AmlCompute status, use get_status()
    print(compute_target.get_status().serialize())
```

### B.2.5 Download TFG dataset

```
from azureml.core import Workspace, Dataset

workspace = Workspace(subscription_id, resource_group, workspace_name)

_overwrite = True
load_data_in_local = False

if load_data_in_local:
    data_folder = os.path.join(os.getcwd(), 'dataset/384x384')
    os.makedirs(data_folder, exist_ok=True)
    img_train_path = os.path.join(data_folder, 'img_train')
    ann_train_path = os.path.join(data_folder, 'ann_train')
    img_val_path = os.path.join(data_folder, 'img_val')
    ann_val_path = os.path.join(data_folder, 'ann_val')
    img_test_path = os.path.join(data_folder, 'img_test')
    ann_test_path = os.path.join(data_folder, 'ann_test')

    img_train.download(target_path=img_train_path, overwrite=_overwrite)
    ann_train.download(target_path=ann_train_path, overwrite=_overwrite)
    img_val.download(target_path=img_val_path, overwrite=_overwrite)
    ann_val.download(target_path=ann_val_path, overwrite=_overwrite)
    img_test.download(target_path=img_test_path, overwrite=_overwrite)
    ann_test.download(target_path=ann_test_path, overwrite=_overwrite)
    print('Loaded data in local')

else:
    img_train = Dataset.get_by_name(
        workspace, name=storage_iput_prefix+'_train_img')
    ann_train = Dataset.get_by_name(
        workspace, name=storage_iput_prefix+'_train_ann')

    img_val = Dataset.get_by_name(
        workspace, name=storage_iput_prefix+'_val_img')
    ann_val = Dataset.get_by_name(
        workspace, name=storage_iput_prefix+'_val_ann')
    img_test = Dataset.get_by_name(
        workspace, name=storage_iput_prefix+'_test_img')
    ann_test = Dataset.get_by_name(
        workspace, name=storage_iput_prefix+'_test_ann')
```

```
    img_train = img_train.register(workspace=ws,
                                   name='img_train',
                                   description='img_train',
                                   create_new_version=True)
    ann_train = ann_train.register(workspace=ws,
                                   name='ann_train',
                                   description='ann_train',
                                   create_new_version=True)
    img_val = img_val.register(workspace=ws,
                               name='img_val',
                               description='img_val',
                               create_new_version=True)
    ann_val = ann_val.register(workspace=ws,
                               name='ann_val',
                               description='ann_val',
                               create_new_version=True)
    img_test = img_test.register(workspace=ws,
                                 name='img_test',
                                 description='img_test',
                                 create_new_version=True)
    ann_test = ann_test.register(workspace=ws,
                                 name='ann_test',
                                 description='ann_test',
                                 create_new_version=True)
    print('Merged specified data into generic experiment dataset container')

import os
script_folder = os.path.join(os.getcwd(), experiment_folder_name)
os.makedirs(script_folder, exist_ok=True)
```

### B.2.6 Create a training script

To submit the job to the cluster, first create a training script. Run the following code to create the training script called *train.py* in the directory you just created.

```
%%writefile $script_folder/train.py

import numpy as np
import os
import joblib
import utils as my_utils
from azureml.core import Run
from keras_segmentation.models.unet import *
from keras_segmentation.models.fcn import *
from keras_segmentation.data_utils.data_loader import image_segmentation_generat
or
from keras_segmentation.predict import evaluate


# get hold of the current run
```

```
run = Run.get_context()

# get experiment inputs
args = my_utils.azure_get_experiment_inputs()

architecture_name = args.architecture_name
model_name = args.model_name

n_classes = args.n_classes
img_train = args.img_train
ann_train = args.ann_train
img_val = args.img_val
ann_val = args.ann_val
img_test = args.img_test
ann_test = args.ann_test
batch_size = args.batch_size
steps_per_epoch = args.steps_per_epoch
val_batch_size = args.val_batch_size
val_steps_per_epoch = args.val_steps_per_epoch
epochs = args.epochs
image_size = args.image_size
optimizer_name = args.optimizer_name

# define path to store checkpoints
os.makedirs('outputs/checkpoint', exist_ok=True)
checkpoint_path = 'outputs/checkpoint/'+model_name+'_1'

# TRAIN
if(architecture_name == 'FCN-8'):
    if(model_name == 'VGG-16'):
        model = fcn_8_vgg(n_classes=n_classes,
                          input_height=image_size, input_width=image_size)
    elif(model_name == 'Resnet-50'):
        model = fcn_8_resnet50(n_classes=n_classes,
                               input_height=image_size, input_width=image_size)
    elif(model_name == 'MobileNet'):
        model = fcn_8_mobilenet(n_classes=n_classes,
                                input_height=image_size, input_width=image_size)
    else:
        raise Exception('Sorry, architecture name: {} no contains model name: {}
').format(
            architecture_name, model_name)

elif(architecture_name == 'UNET'):
    if(model_name == 'VGG-16'):
        model = vgg_unet(n_classes=n_classes,
                         input_height=image_size, input_width=image_size)
    elif(model_name == 'Resnet-50'):
        model = resnet50_unet(n_classes=n_classes,
                              input_height=image_size, input_width=image_size)
    elif(model_name == 'MobileNet'):
        model = mobilenet_unet(n_classes=n_classes,
                               input_height=image_size, input_width=image_size)
```

```
    else:
        raise Exception('Sorry, architecture name: {} no contains model name: {}
').format(
            architecture_name, model_name)
else:
    raise Exception('Sorry, architecture name: {} not found').format(
        architecture_name)

print('Selected train model is '+model_name +' & architecture is '+architecture_
name)
model.summary()

model.train(
    train_images=img_train,
    train_annotations=ann_train,
    val_images=img_val,
    val_annotations=ann_val,
    checkpoints_path=checkpoint_path,
    validate=True,
    batch_size=batch_size,
    val_batch_size=val_batch_size,
    steps_per_epoch=steps_per_epoch,
    val_steps_per_epoch=val_steps_per_epoch,
    epochs=epochs,
    optimizer_name=optimizer_name
)

my_utils.save_checkpoints_in_zip()

my_utils.azure_log_train_accuracy(model, run)
my_utils.azure_log_plot_train_accuracy(model, run)


# TEST
evaluation = evaluate(model=model, inp_images_dir=img_test,
                      annotations_dir=ann_test)

my_utils.azure_log_test_data(evaluation, run)

# Save model, the outputs folder is automatically uploaded into experiment recor
d by AML Compute
model.save('./outputs/model.h5')
joblib.dump(value=model, filename='outputs/execution_model.pkl')
```

### B.2.7 Create a utils script

Create a utils script *utils.py* to add support functionalities about train script.

```
%%writefile $script_folder/utils.py

import glob
```

```
import joblib
import zipfile
import os
import argparse
import numpy as np
import matplotlib.pyplot as plt

def azure_get_experiment_inputs():

    parser = argparse.ArgumentParser()

    #dataset parser
    parser.add_argument('--img_train', type=str,
                        dest='img_train', help='img_train')
    parser.add_argument('--ann_train', type=str,
                        dest='ann_train', help='ann_train')
    parser.add_argument('--img_val', type=str, dest='img_val', help='img_val')
    parser.add_argument('--ann_val', type=str, dest='ann_val', help='ann_val')
    parser.add_argument('--img_test', type=str,
                        dest='img_test', help='img_test')
    parser.add_argument('--ann_test', type=str,
                        dest='ann_test', help='ann_test')
    #train parser
    parser.add_argument('--architecture_name', type=str,
                        dest='architecture_name', help='architecture_name')
    parser.add_argument('--model_name', type=str,
                        dest='model_name', help='model_name')
    parser.add_argument('--n_classes', type=int,
                        dest='n_classes', default=3, help='n_classes')
    parser.add_argument('--batch_size', type=int,
                        dest='batch_size', default=10, help='batch size')
    parser.add_argument('--steps_per_epoch', type=int,
                        dest='steps_per_epoch', default=1, help='steps per epoch
')
    parser.add_argument('--val_batch_size', type=int,
                        dest='val_batch_size', default=10, help='val batch size'
)
    parser.add_argument('--val_steps_per_epoch', type=int,
                        dest='val_steps_per_epoch', default=1, help='val steps p
er epoch')
    parser.add_argument('--epochs', type=int, dest='epochs',
                        default=5, help='epochs')
    parser.add_argument('--image_size', type=int,
                        dest='image_size', default=224, help='image size')
    parser.add_argument('--optimizer_name', type=str,
                        dest='optimizer_name', default="adadelta", help='optimiz
er name')

    return parser.parse_args()


def save_checkpoints_in_zip():
```

```
    path = os.path.join(os.getcwd(), 'outputs')
    path = os.path.abspath(os.path.normpath(os.path.expanduser(path)))
    for folder in os.listdir(path):
        zipf = zipfile.ZipFile('{0}.zip'.format(
            os.path.join(path, folder)), 'w', zipfile.ZIP_DEFLATED)
        for root, dirs, files in os.walk(os.path.join(path, folder)):
            for filename in files:
                zipf.write(os.path.abspath(os.path.join(
                    root, filename)), arcname=filename)
        zipf.close()

def azure_log_train_accuracy(model, run):
    run.log_list('Train accuracy -backup',
                 model.history.history['accuracy'][:25], description='TRAIN ACCU
RACY')
    run.log_list('Train accuracy -backup',
                 model.history.history['accuracy'][25:50], description='TRAIN AC
CURACY')
    run.log_list('Train accuracy -backup',
                 model.history.history['accuracy'][50:75], description='TRAIN AC
CURACY')
    run.log_list('Train accuracy -backup',
                 model.history.history['accuracy'][75:], description='TRAIN ACCU
RACY')

    run.log_list('Validation accuracy -backup',
                 model.history.history['val_accuracy'][:25], description='VALIDA
TION ACCURACY')
    run.log_list('Validation accuracy -backup',
                 model.history.history['val_accuracy'][25:50], description='VALI
DATION ACCURACY')
    run.log_list('Validation accuracy -backup',
                 model.history.history['val_accuracy'][50:75], description='VALI
DATION ACCURACY')
    run.log_list('Validation accuracy -backup',
                 model.history.history['val_accuracy'][75:], description='VALIDA
TION ACCURACY')

    run.log_list('Train loss -backup',
                 model.history.history['loss'][:25], description='TRAIN LOSS')
    run.log_list('Train loss -backup',
                 model.history.history['loss'][25:50], description='TRAIN LOSS')
    run.log_list('Train loss -backup',
                 model.history.history['loss'][50:75], description='TRAIN LOSS')
    run.log_list('Train loss -backup',
                 model.history.history['loss'][75:], description='TRAIN LOSS')

    run.log_list('Validation loss -backup',
                 model.history.history['val_loss'][:25], description='VALIDATION
LOSS')
    run.log_list('Validation loss -backup',
                 model.history.history['val_loss'][25:50], description='VALIDATI
ON LOSS')
```

```
    run.log_list('Validation loss -backup',
                 model.history.history['val_loss'][50:75], description='VALIDATI
ON LOSS')
    run.log_list('Validation loss -backup',
                 model.history.history['val_loss'][75:], description='VALIDATION
LOSS')

    run.log("final_val_loss", model.history.history["val_loss"][-1])


def azure_log_plot_train_accuracy(model, run):
    plt.figure(0)
    plt.plot(model.history.history['accuracy'], 'b', label="accuracy")
    plt.plot(model.history.history['val_accuracy'], 'g', label="val_accuracy")
    plt.xticks(np.arange(0, 101, 5))
    plt.yticks(np.arange(0, 1, 0.05))
    plt.xlabel("Num of epochs")
    plt.ylabel("Accuracy")
    plt.title("Training accuracy vs Validation accuracy")
    plt.legend(['train', 'validation'])
    plt.grid(True)
    run.log_image("accuracy vs val_accuracy", plot=plt)

    plt.figure(1)
    plt.plot(model.history.history['loss'], 'b')
    plt.plot(model.history.history['val_loss'], 'g')
    plt.xticks(np.arange(0, 101, 5))
    #plt.yticks(np.arange(0, 10, 0.2))
    plt.xlabel("Num of epochs")
    plt.ylabel("Loss")
    plt.title("Training loss vs Validation loss")
    plt.legend(['train', 'validation'])
    plt.grid(True)
    run.log_image("training loss vs validation loss", plot=plt)


def azure_log_test_data(evaluation, run):
    run.log('frequency_weighted_IU', evaluation['frequency_weighted_IU'])
    run.log('mean_IU', evaluation['mean_IU'])
    run.log('Ground mean_IoU ', evaluation['class_wise_IU'][0])
    run.log('Tree mean_IoU ', evaluation['class_wise_IU'][1])
    run.log('Grass mean_IoU ', evaluation['class_wise_IU'][2])
    run.log_list('class_wise_IU', evaluation['class_wise_IU'])
```

### B.2.8 Define script params

Next, we define script params to pass our defined train inputs

```
script_params = {
    '--n_classes': n_classes,
    '--architecture_name': architecture_name,
    '--model_name': model_name,
```

```
    '--img_train': img_train.as_named_input('img_train').as_mount(),
    '--ann_train': ann_train.as_named_input('ann_train').as_mount(),
    '--img_val': img_val.as_named_input('img_val').as_mount(),
    '--ann_val': ann_val.as_named_input('ann_val').as_mount(),
    '--img_test': img_test.as_named_input('img_test').as_mount(),
    '--ann_test': ann_test.as_named_input('ann_test').as_mount(),
    '--batch_size': batch_size,
    '--val_batch_size': val_batch_size,
    '--steps_per_epoch': steps_per_epoch,
    '--val_steps_per_epoch': val_steps_per_epoch,
    '--epochs': epochs,
    '--image_size': image_size
}
```

### B.2.9 Define Tensorflow estimator & add dependencies

Define a Tensorflow context to execute our experiment (*train.py*) over GPU using train input specifications (*script_params*) and defined dependencies.

```
from azureml.train.dnn import TensorFlow
est = TensorFlow(source_directory=script_folder,
                 entry_script='train.py',
                 script_params=script_params,
                 compute_target=compute_target,
                 pip_packages=[
                     "keras",
                     "imageio",
                     "imgaug",
                     "opencv-python",
                     "tqdm",
                     "joblib",
                     "matplotlib"],
                 use_gpu=True)
```

### B.2.10 Execute individual experiment

```
if(single_experiment):
    run = exp.submit(est)
```

## B3. Improve experiment executions

### B.3.1 Hyperparameters tunning

Define tune params to test and experiment which combinations gets best performance. In this case, it will be used to make multiple experiment sequences using a different model for each iteration.

Also, we can use a ´Grid Parameter Sampling´ to find best batch_size combination for each model and architecture.

```
from azureml.train.hyperdrive import RandomParameterSampling, GridParameterSampl
ing
from azureml.train.hyperdrive import choice

find_best_batch_size = False

if (find_best_batch_size):
    ps = GridParameterSampling({
        '--batch_size': choice(140, 112, 80, 70, 56, 40, 35),
        '--architecture_name': choice('FCN-8', 'UNET'),
        '--model_name': choice('VGG-16', 'Resnet-50', ''),
        #   560,280,140,112,80,70,56,40,35,28,20,16,14,10,8,7,5,4
        '--steps_per_epoch': choice(1)
    })
else:
    ps = RandomParameterSampling(
        {
            '--model_name': choice('VGG-16', 'Resnet-50'),
            '--architecture_name': choice('FCN-8', 'UNET', 'MobileNet')
            # '--optimizer_name': choice("SGD", "RMSprop", "Adam", "Adadelta", "
Adagrad", "Adamax", "Nadam")
        }
    )
```

### B.3.2 Stopper policies

Define a stopper policies to perform our time execution and improve our financial expense

*Median stopping* is an early termination policy based on running averages of primary metrics reported by the runs. This policy computes running averages across all training runs and terminates runs whose performance is worse than the median of the running averages.

```
from azureml.train.hyperdrive import BanditPolicy, MedianStoppingPolicy

#policy = BanditPolicy(evaluation_interval=50, slack_factor=0.1)
policy = MedianStoppingPolicy(evaluation_interval=1, delay_evaluation=20)
```

## B4. Release and validate result

### B.4.1 Execute experiment

Finally, we will define and hypedrive context and will execute it.

```
from azureml.train.hyperdrive import HyperDriveConfig, PrimaryMetricGoal

hdc = HyperDriveConfig(estimator=est,
                       hyperparameter_sampling=ps,
                       policy=policy,
```

```
                    primary_metric_name='Validation accuracy',
                    primary_metric_goal=PrimaryMetricGoal.MAXIMIZE,
                    max_total_runs=200,
                    max_concurrent_runs=1)


hdr = exp.submit(config=hdc)
```

### B.4.2 Display run results

You now have a model trained on a remote cluster. Retrieve all the metrics logged during the run, including the accuracy of the model:

```
from azureml.widgets import RunDetails
RunDetails(hdr).show()
```

### B.4.3 Check best result and best params combination

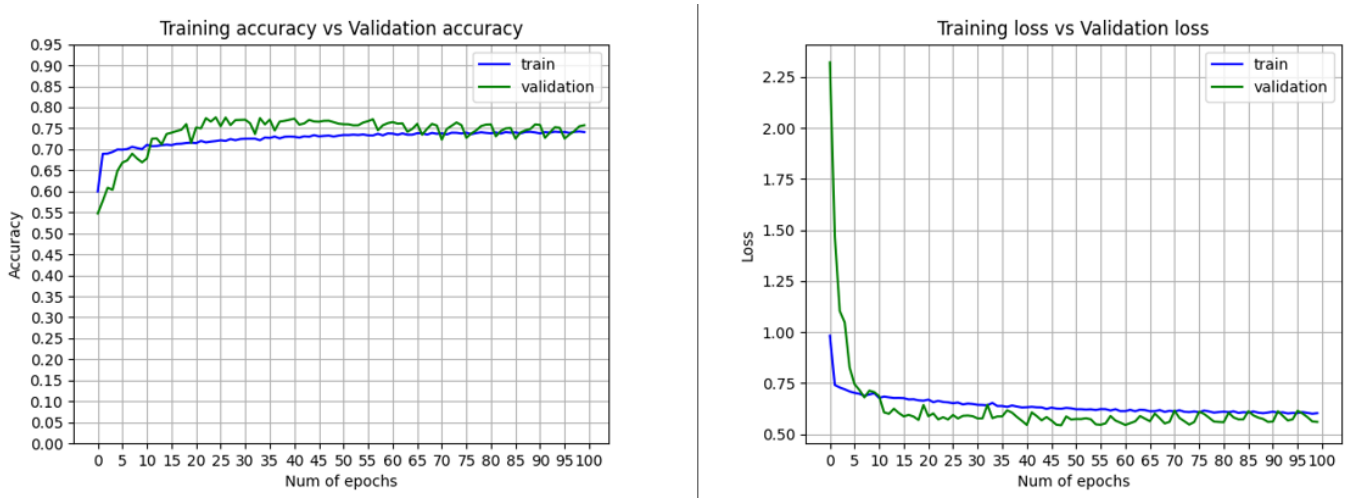
```
hdr.wait_for_completion(show_output=True)
assert(hdr.get_status() == "Completed")
best_run = hdr.get_best_run_by_primary_metric()
print(best_run.get_details()['runDefinition']['arguments'])
```

# Annex C - Resultat dels entrenaments

A continuació s'adjunta els resultats dels entrenaments realitzats que ens permet conèixer i avaluar com de bo és el nostre model i si aquest generalitza correctament.

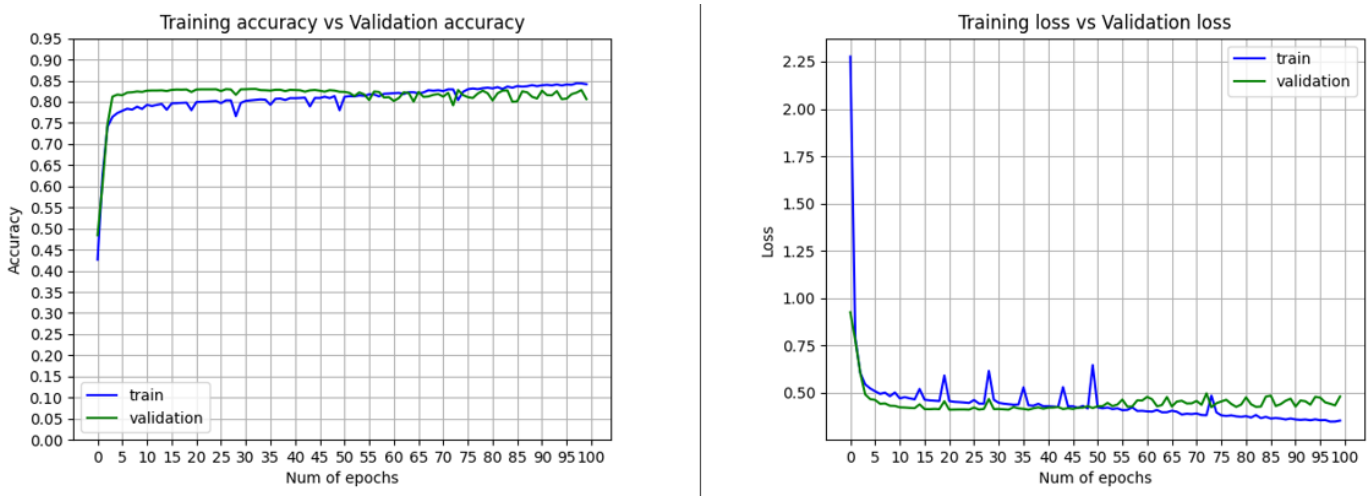
## C1. VGG-16

- FCN-8
  - Imatge 224x224



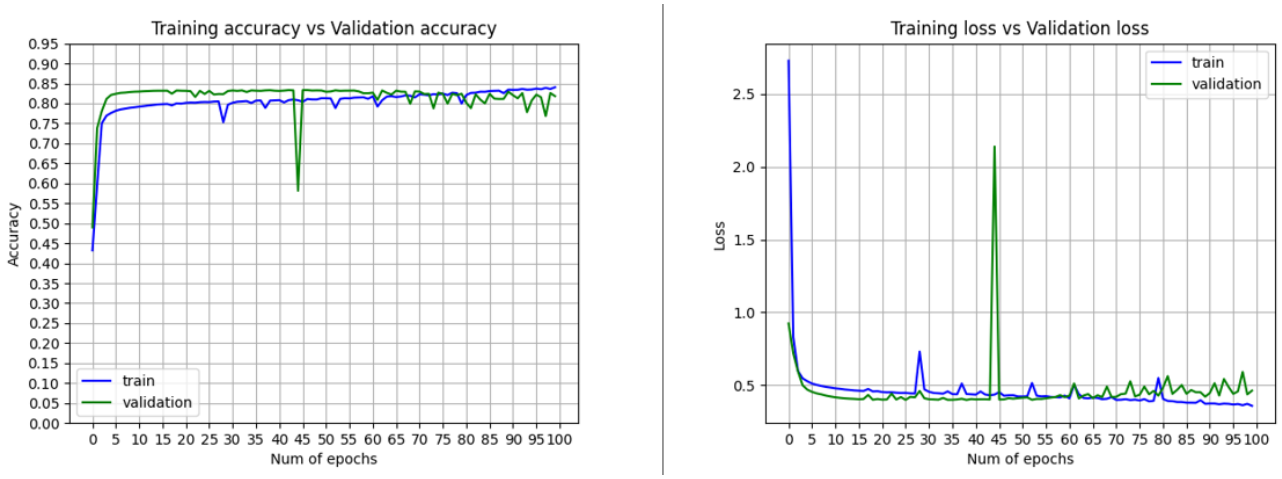

*Il·lustració 42. Resultat entrenament VGG-16 FCN-8 224x224*

  - Imatge 384x384



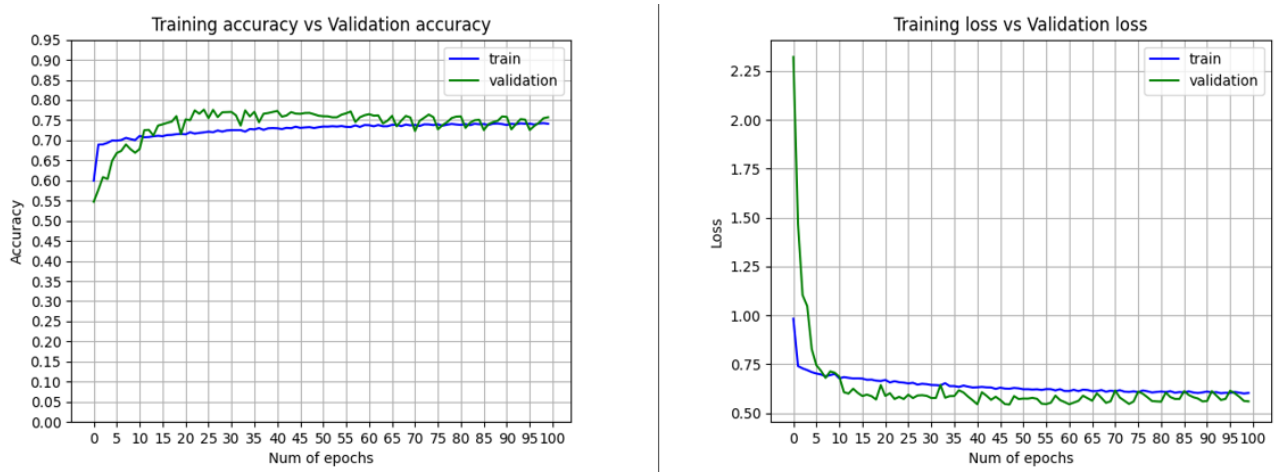

*Il·lustració 43. Resultat entrenament VGG-16 FCN-8 384x384*

- Imatge 512x512



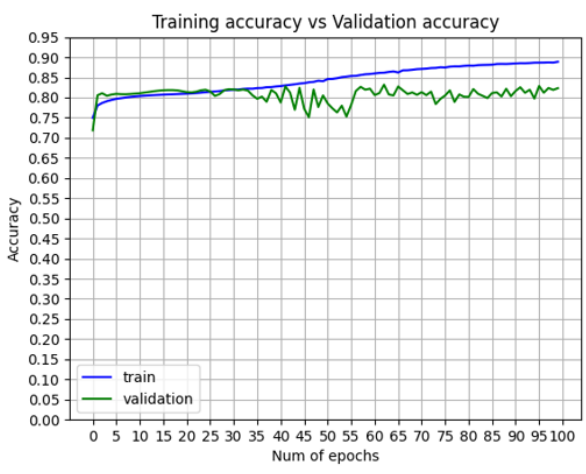

*Il·lustració 44. Resultat entrenament VGG-16 UNET 512x512*

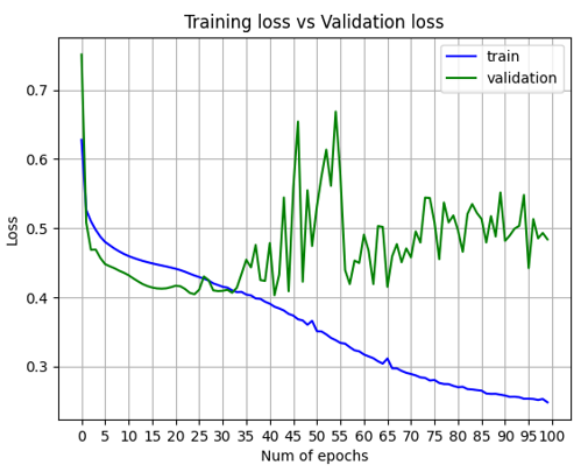
- UNET
  - Imatge 224x224




*Il·lustració 45. Resultat entrenament VGG-16 UNET 224x224*

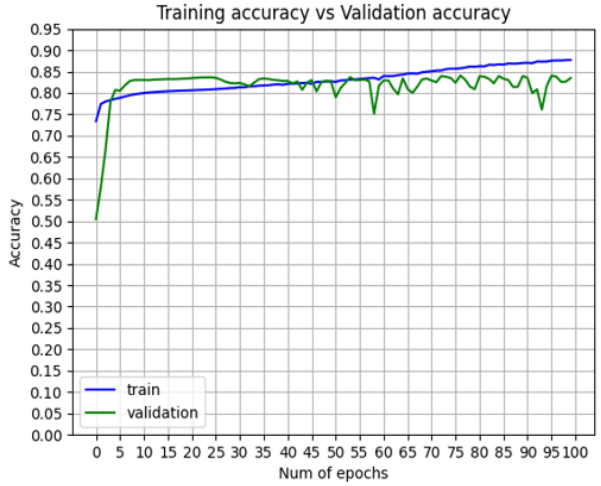
- Imatge 384x384

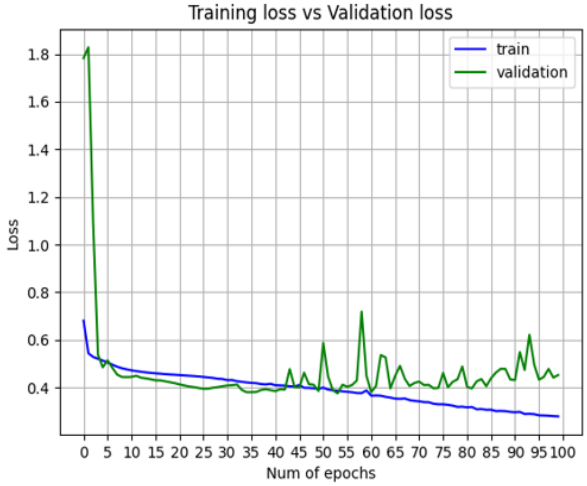



*Il·lustració 46Il·lustració 42. Resultat entrenament VGG-16 UNET 384x384*

- Imatge 512x512

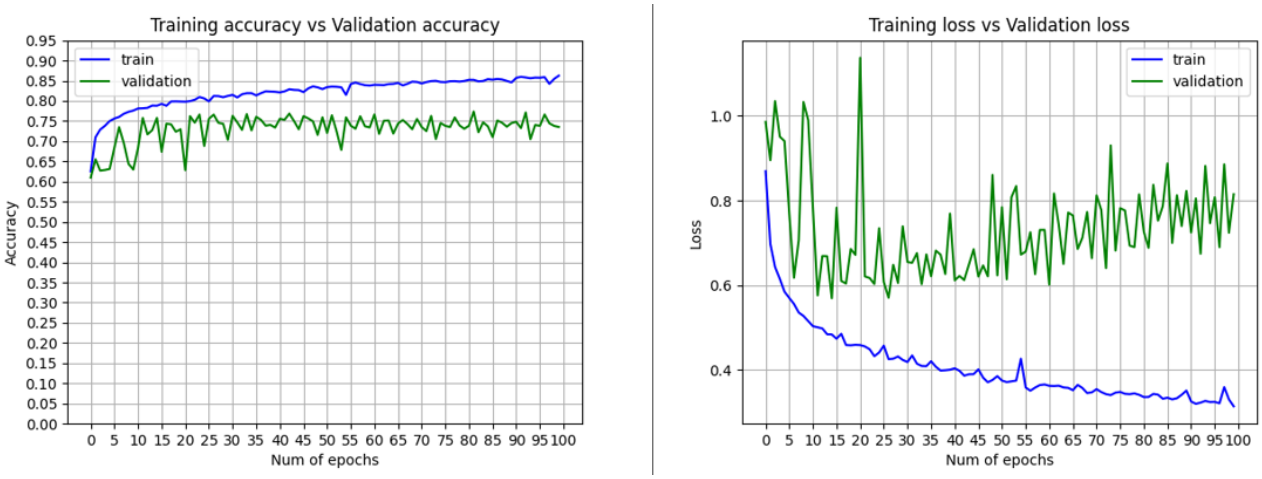



*Il·lustració 47. Resultat entrenament VGG-16 UNET 512x512*

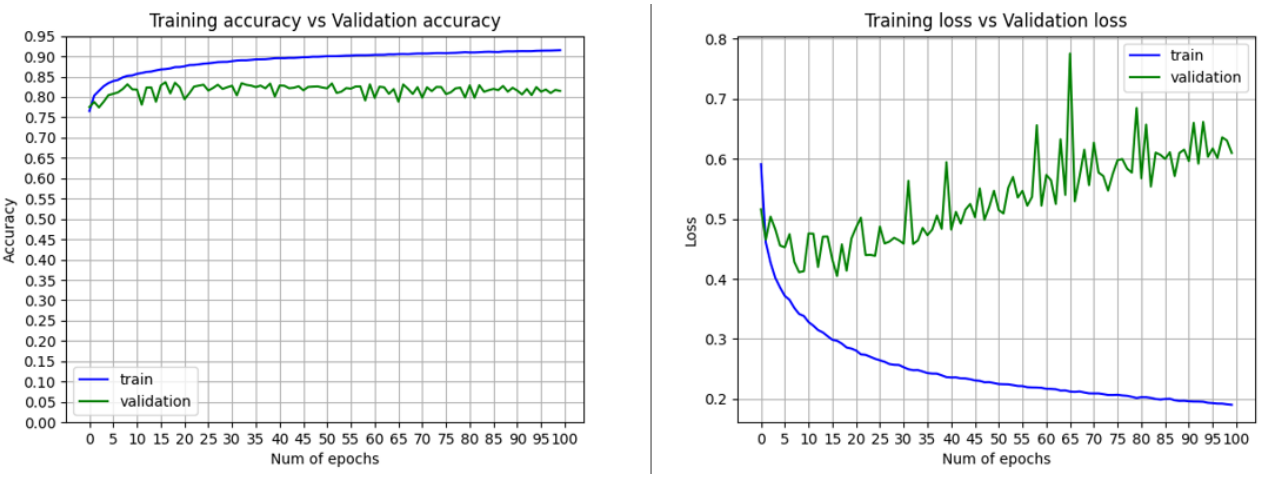
## C2. Resnet-50

- UNET
  - Imatge 224x224




*Il·lustració 48. Resultat entrenament Resnet UNET 224x224*

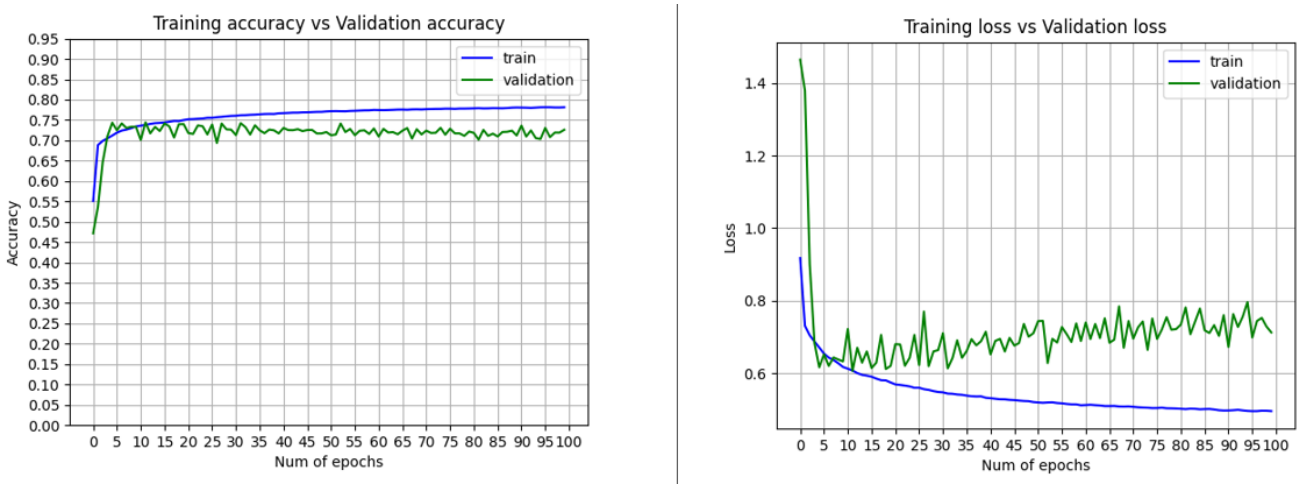
  - Imatge 384x384




*Il·lustració 49. Resultat entrenament Resnet UNET 384X384*

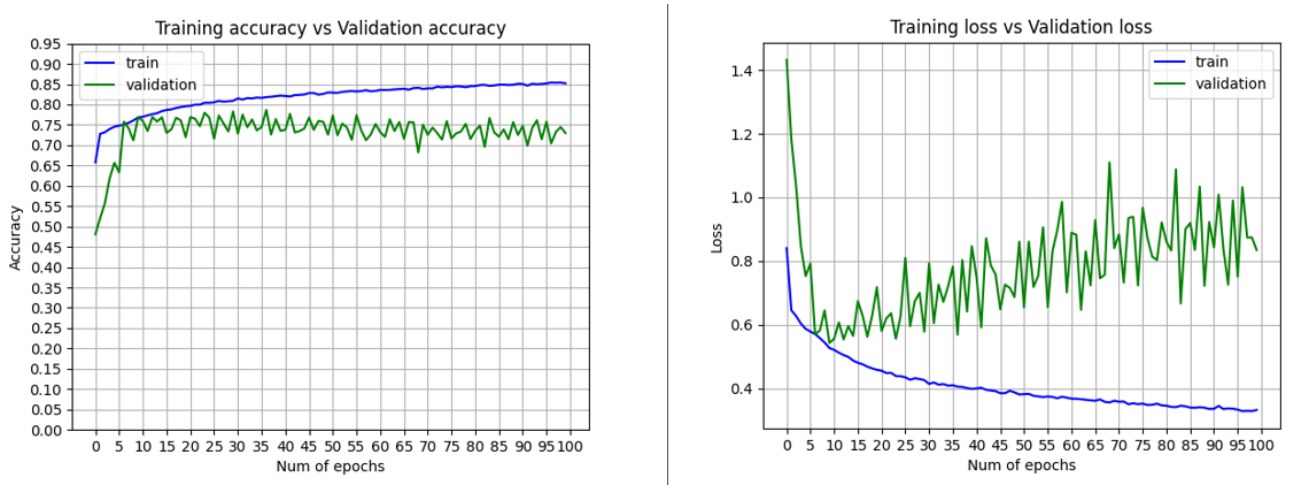
## C3. MobileNet

- FCN-8
  - Imatge 224x224




*Il·lustració 50. Resultat entrenament MobileNet FCN-8 224x224*

- UNET
  - Imatge 224x224




*Il·lustració 51. Resultat entrenament MobileNet UNET 224x224*