



Treball de Fi de Grau

GRAU D'ENGINYERIA INFORMÀTICA

**Facultat de Matemàtiques i Informàtica
Universitat de Barcelona**

**SISTEMA DE DETECCIÓ I CLASSIFICACIÓ
DE SENYALS DE TRÀNSIT BASAT EN YOLO**

Joan Viñas Simon

Director: Dra. Laura Igual Muñoz
Realitzat a: Departament de
Matemàtiques i Informàtica
Barcelona, 15 de gener del 2025

Resum

Aquest és un treball enfocat a entendre el funcionament de les xarxes neuronals i les seves aplicacions en el camp de la visió artificial. Conèixer en detall com un model, prèviament entrenat, pot detectar i classificar elements d'una imatge de manera precisa i eficient. Amb aquesta finalitat, s'avaluarà els resultats obtinguts d'un sistema de detecció i classificació de senyals de trànsit.

La investigació es dirigeix a explorar com un model de xarxa neuronal convolucional pot extreure informació rellevant d'una imatge per reconèixer objectes i assignar-los a una classe específica. Això inclou tant la comprensió de les tècniques bàsiques de les xarxes neuronals com les arquitectures més avançades, les xarxes convolucionals, que són fonamentals per garantir l'eficàcia dels sistemes de detecció.

L'algorisme amb el qual treballarem és el YOLO (*You Only Look Once*), conegut per la seva capacitat de realitzar detecció multiclasse en temps real. Es realitzarà una sèrie d'experiments per tal de posar en pràctica els coneixements adquirits, utilitzant dos conjunts de dades d'imatges de senyals de trànsit, un corresponent a Europa i l'altre únicament d'Alemanya.

Finalment, intentarem millorar i avaluar els resultats obtinguts a través d'estratègies com el *fine-tuning* i l'anàlisi de les mètriques obtingudes com la corba ROC, precisió i *recall*, entre d'altres.

Resumen

Este trabajo se centra en comprender el funcionamiento de las redes neuronales y sus aplicaciones en el campo de la visión artificial. En concreto, busca explorar en detalle cómo un modelo previamente entrenado puede detectar y clasificar elementos de una imagen de manera precisa y eficiente. Con este objetivo, evaluaremos los resultados obtenidos de un sistema de detección y clasificación de señales de tráfico.

La investigación tiene como propósito explorar cómo un modelo de red neuronal convolucional puede extraer información relevante de una imagen para reconocer objetos y asignarlos a una clase específica. Esto incluye tanto la comprensión de las técnicas básicas de redes neuronales como de arquitecturas más avanzadas, como las redes convolucionales, que son fundamentales para garantizar la eficacia de los sistemas de detección.

El algoritmo con el que trabajaremos es YOLO (*You Only Look Once*), conocido por su capacidad para realizar detección multiclase en tiempo real. Se llevarán a cabo una serie de experimentos para aplicar los conocimientos adquiridos, utilizando dos conjuntos de datos de imágenes de señales de tráfico: uno correspondiente a Europa y otro exclusivamente de Alemania.

Finalmente, trataremos de mejorar y evaluar los resultados obtenidos mediante estrategias como el *fine-tuning* y el análisis de métricas como la curva ROC, la precisión y el recall, entre otras.

Abstract

The main focus of this assignment is the understanding of how neural networks work and their applications in the field of computer vision. Specifically, it aims to explore in detail how a pre-trained model can accurately and efficiently detect and classify elements within an image. For this purpose, we will evaluate the results obtained from a traffic sign detection and classification system.

The research seeks to investigate how a convolutional neural network model can extract relevant information from an image to recognize objects and assign them to a specific class. This includes understanding both the fundamental techniques of neural networks and the more advanced architectures, such as CNNs, which are critical to ensuring the effectiveness of detection systems.

The algorithm we will work with is YOLO (*You Only Look Once*), known for its ability to perform multi-class detection in real time. A series of experiments will be conducted to apply the knowledge acquired, using two datasets of traffic sign images: one corresponding to Europe and the other exclusively from Germany.

Finally, we will aim to improve and evaluate the results obtained through strategies such as *fine-tuning* and the analysis of metrics such as the ROC curve, precision, and recall, among others.

Índex

1.	Introducció	7
1.1.	Context i objectius del treball	7
1.2.	Planificació del treball	8
2.	Intel·ligència Artificial	10
2.1.	Introducció	10
2.2.	Machine Learning	10
2.3.	Deep Learning	11
2.4.	Xarxes Neuronals	12
2.5.	Xarxes Neuronals Convolucionals	14
2.5.1.	Capa Convolucional	15
2.5.2.	Capa Pooling	17
2.5.3.	Capa totalment connectada	18
3.	Models de detecció i classificació d'objectes	19
3.1.	Què és la detecció i la classificació?	19
3.2.	YOLO (You Only Look Once)	20
3.2.1.	Introducció	20
3.2.2.	Arquitectura	21
3.2.3.	Funcionament del model	21
3.2.4.	Detecció multiclasse	23
3.3.	Alternatives a YOLO	28
4.	Entrenament del model i fine-tuning	32
5.	Llibreries i recursos computacionals	35
5.1.	Ultralytics	35
5.2.	Google Colab.....	35
6.	Conjunts de dades	36
6.1.	Introducció.....	36
6.2.	EuRoSign.....	36
6.3.	GTSDb - German Traffic Sign Detection Benchmark	39
6.4.	COCO	42
6.5.	Preparació dels conjunts	43
7.	Mètriques d'avaluació.....	46
8.	Experiments i resultats.....	53
8.1.	Model A	53
8.2.	Model B	54
8.3.	Model C	54
8.4.	Resultats Qualitatius	54
8.5.	Resultats Quantitatius	79

8.6.	Comparativa de resultat quantitatiu i qualitatiu	87
8.7.	Inferència sobre video en temps real	88
9.	Conclusions del treball	89
9.1.	Problemes sorgits	89
9.1.1.	CPU i GPU	89
9.1.2.	Avaluació de diferents classes	90
9.1.3.	Conjunts de dades similars	90
9.2.	Conclusions	91
9.3.	Futures línies d'investigació	91
10.	Bibliografia	92
11.	Annex.....	95
11.1.	Notebook de preparació pel conjunt EuRoSign.....	95
11.2.	Notebook de preparació pel conjunt GTSDb.....	97
11.3.	Notebook de treball dels models.....	101

1. Introducció

Durant els meus estudis en Enginyeria Informàtica, he tingut l'oportunitat de treballar en diversos projectes i tecnologies. No obstant això, un dels camps que havia explorat poc era el *deep learning*, pel que el meu coneixement sobre el tema era limitat.

Vaig tenir la sort de fer les pràctiques del grau en una empresa que es dedica a optimitzar processos de piscifactories mitjançant intel·ligència artificial. Una de les eines que utilitzaven era YOLO, per a una tasca de segmentació.

Des d'un inici vaig tenir interès per aquesta tecnologia, que desconeixia completament. Al veure als meus companys treballar amb ella, vaig tenir ganes d'aprendre'n per poder-los ajudar i participar activament en el projecte. Per això també he volgut realitzar el treball sobre aquest model.

Amb l'ajuda de la professora del treball, a qui personalment li estic molt agraït, vam trobar un àmbit quotidià on es podria aplicar el model i avaluar-lo.

1.1. Context i objectius del treball

Amb la intel·ligència artificial cada vegada més integrada en la nostra vida quotidiana, són molts els sistemes intel·ligents que trobem diàriament al nostre voltant. Des del reconeixement facial en dispositius mòbils fins als sistemes de detecció d'objectes en vehicles, la visió artificial ha demostrat ser una tecnologia clau en molts àmbits.

En aquest treball ens centrarem en una aplicació concreta d'aquesta disciplina: la implementació d'un sistema capaç de detectar i classificar senyals de trànsit presents en imatges.

Un dels sectors amb un creixement més destacat és el dels vehicles de conducció autònoma, els quals fan ús d'aquests sistemes per orientar-se i navegar amb seguretat. Aquests vehicles han de ser capaços d'identificar ràpidament elements clau del seu entorn, com ara senyals de trànsit, vianants o altres vehicles, per prendre decisions en fraccions de segon.

És per això que neix YOLO. Un model dissenyat per fer prediccions a velocitats, fins aleshores inèdites, gràcies a la forma pionera en com realitza la detecció i classificació dels objectes.

Es realitzarà una sèrie d'experiments amb aquest model, entrenat amb diferents conjunts de dades amb l'objectiu d'aconseguir el millor sistema de detecció i classificació.

Aquest projecte ens permetrà explorar diferents etapes del flux de treball en *deep learning*, com ara la preparació de dades, l'entrenament de models, l'optimització d'hiperparàmetres i l'avaluació de resultats.



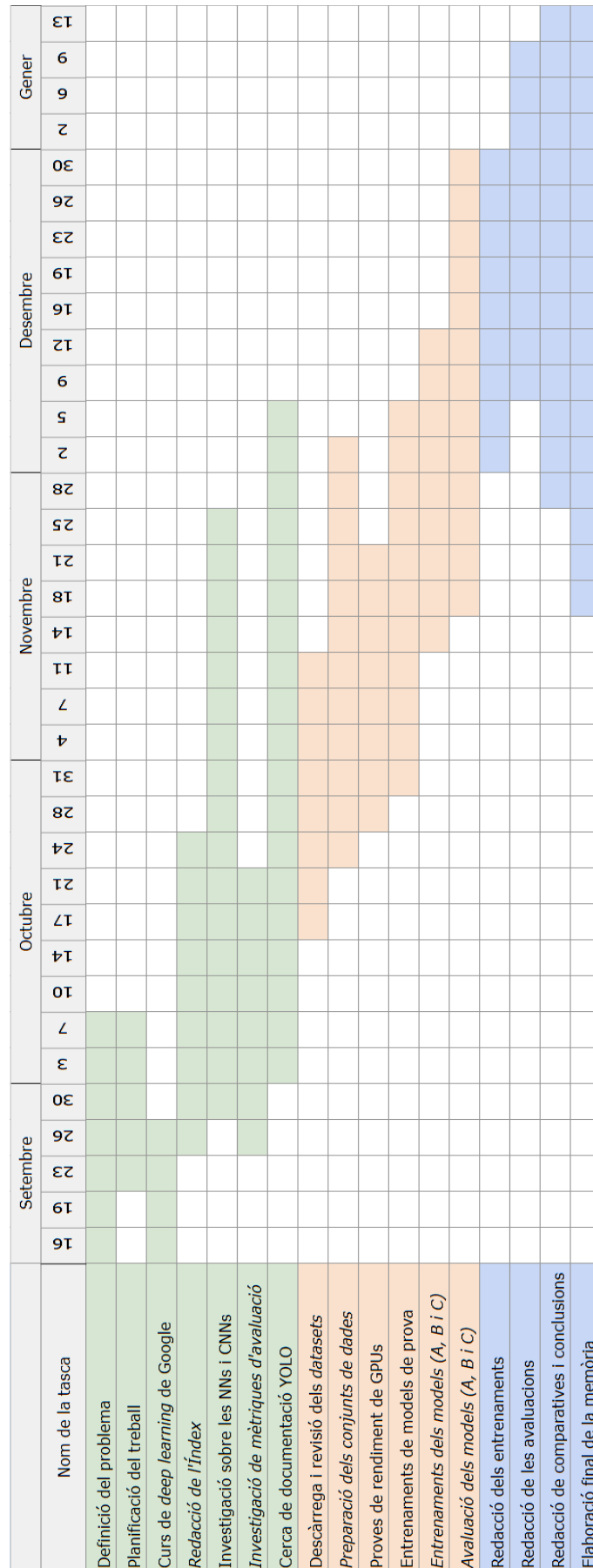
Figura 1. Detecció i classificació d'una mostra amb el model YOLO

1.2. Planificació del treball

El pla de treball del TFG s'ha organitzat d'acord amb el calendari acadèmic del primer semestre del curs 2024/2025, amb data de finalització establerta pel dia 15 de gener de 2025.

Si es va començar el treball el dia 16 de setembre del 2024, he disposat d'un període total de quatre mesos per a la seva realització.

Amb el següent diagrama de Gantt (*taula 1*), es presenta el desglossament detallat de les tasques desenvolupades durant aquest temps.



Taula 1. Diagrama de Gantt de la planificació del TFG

2. Intel·ligència Artificial

2.1. Introducció

El concepte d'Intel·ligència artificial (AI) està definit pel Parlament Europeu com “la capacitat d'una màquina de presentar les mateixes habilitats que els éssers humans, com el raonament, l'aprenentatge, la creativitat i la capacitat de planificar”.

Tot això amb l'objectiu de realitzar tasques intel·lectuals normalment realitzades pels humans. Aquestes tasques poden ser del tot diverses, és per això que el concepte d'IA s'estructura en diferents branques. Aquest treball se centra específicament en la branca de *Machine Learning*.

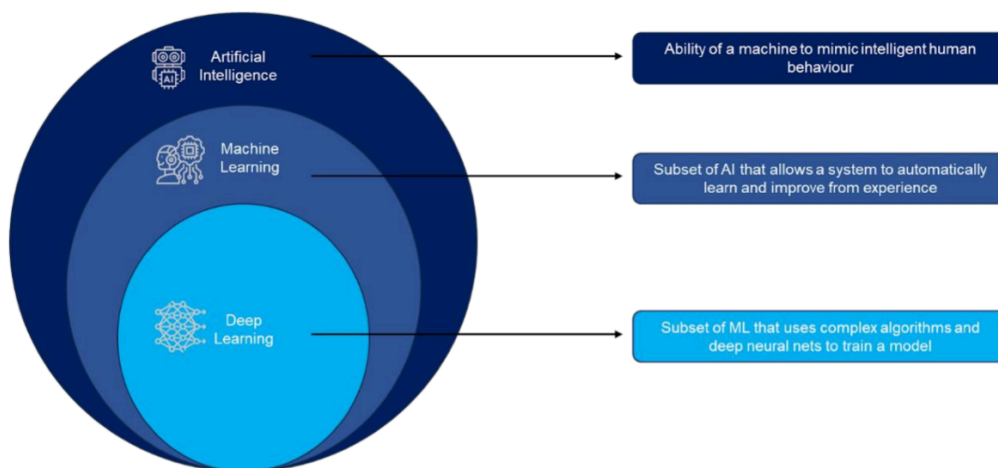


Figura 2. Esquema dels nivells conceptuals d'AI

2.2. Machine Learning

Machine Learning és una disciplina de la Intel·ligència Artificial dedicada a l'aprenentatge automàtic, on, mitjançant algorismes, es dota els ordinadors de la capacitat d'identificar patrons en grans volums de dades i elaborar prediccions.

L'algorisme avalua aquestes prediccions i ajusta els paràmetres del model en cada iteració, millorant-ne el rendiment progressivament, i donant la sensació d'aprenentatge.

Normalment els algorismes de ML es poden classificar en les següents categories:

- **Aprenentatge supervisat:**

En l'aprenentatge supervisat, l'algorisme ha de tenir en el seu conjunt d'entrenament la solució òptima etiquetada. L'objectiu és que el model pugui inferir la relació entre les entrades i les sortides, de manera que pugui predir correctament la sortida per a noves dades d'entrada. Les xarxes neuronals, les màquines de *vector suport* (SVM) i els arbres de decisió són exemples d'algorismes que usen aquest paradigma.

- **Aprenentatge no supervisat:**

Aquests algorismes són entrenats per un conjunt de dades sense etiquetar. El sistema no pot avaluar les prediccions realitzades, i per tant és feina del mateix algoritme trobar patrons a les dades que permetin organitzar-les d'alguna manera. Un exemple d'aprenentatge no supervisat són els algorismes de clustering com el *k-means*.

- **Aprenentatge semi-supervisat:**

Aquests algorismes són entrenats amb dades etiquetades i dades sense etiquetar. En alguns casos s'ha demostrat que l'ús d'aquests algorismes que combinen l'aprenentatge supervisat i el no supervisat resulta en un sistema més precís.

- **Aprenentatge per reforç:**

Aquest algorisme es basa en un agent el qual aprèn a prendre decisions interactuant amb un entorn. Mitjançant un procés de prova i error, l'agent rep recompenses o penalitzacions en funció de les accions que pren, i ajusta la seva estratègia per maximitzar la recompensa total a llarg termini. Aquest enfocament és especialment útil en problemes seqüencials i d'optimització, com el control de robots.

2.3. Deep Learning

El *deep learning* és un tipus d'intel·ligència artificial, derivada del *machine learning*, i és en la que es basa aquest treball. La majoria dels mètodes de *deep learning* utilitzen arquitectures de xarxes neuronals o NNs (*Neural Network*). S'anomenen xarxes neuronals perquè les neurones corresponen a camps receptius d'una manera similar a com ho fan les d'un cervell biològic.

L'aprenentatge és profund perquè les xarxes tenen múltiples capes ocultes, cosa que permet extreure patrons a diferents nivells d'abstracció. Aquesta profunditat és fonamental per assolir una alta precisió en tasques complexes com el reconeixement d'imatges, veu i text.

2.4. Xarxes Neuronals

Una xarxa neuronal està formada per unitats bàsiques anomenades neurones artificials, que s'organitzen en capes.

El valor de sortida d'una xarxa neuronal serà el resultat d'una sèrie de càlculs que comencen a la primera capa, travessen les capes internes i arriben a la capa de sortida. Per a l'usuari només són visibles la capa d'entrada (*Input Layer*) i la capa de sortida (*Output Layer*). Les capes intermèdies es denominen capes ocultes (*Hidden Layers*) perquè són invisibles per l'usuari (*figura 3*).

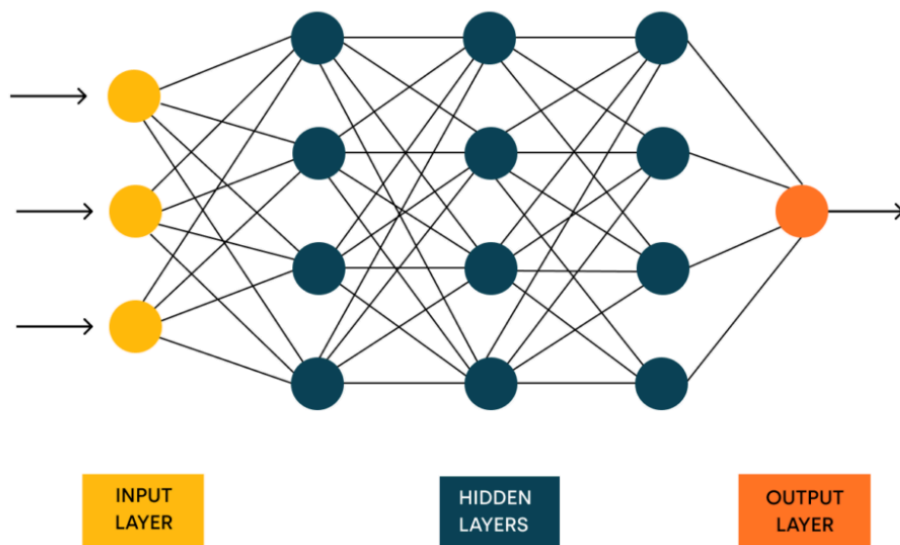


Figura 3. Exemple conceptual d'una xarxa neuronal

Per començar, assignarem un valor a cada neurona de la capa d'entrada. Per exemple, en problemes de reconeixement d'imatges, podem tenir tantes neurones a la primera capa com píxels té una imatge, on a cada neurona se li assignaria la intensitat d'aquell píxel.

Per a les capes ocultes, cada neurona rebrà un valor A que s'anomena activació. Aquest valor és una suma ponderada de totes les neurones de la capa anterior.

$$Z = W_0 A_0 + W_1 A_1 + \dots + W_n A_n + B$$

Observem com en aquesta suma hi apareix un pes W per a cada activació i un biaix B . El pes ens servirà per ponderar l'activació d'una certa neurona amb major o menor importància, i el biaix per ajustar el resultat.

Ara bé, el resultat d'aquesta suma és completament arbitrària. Necessitem una funció, que denominarem **funció d'activació**, per transformar qualsevol valor en un rang comprès entre 0 i 1. Sense funcions d'activació, les xarxes neuronals es limitarien únicament a modelar relacions lineals entre entrades i sortides.

Les funcions d'activació introdueixen no-linealitats, cosa que permet a les xarxes neuronals aprendre patrons més complexos entre entrades i sortides.

Originalment, s'utilitzava la funció **Sigmoid**. Tot i això, en xarxes neuronals modernes, sovint s'usa la funció **ReLU** (*Rectified Linear Unit*) (figura 4). Aquesta funció retorna 0 si l'entrada és negativa, i l'entrada mateixa si és positiva, sent més eficient en termes computacionals.

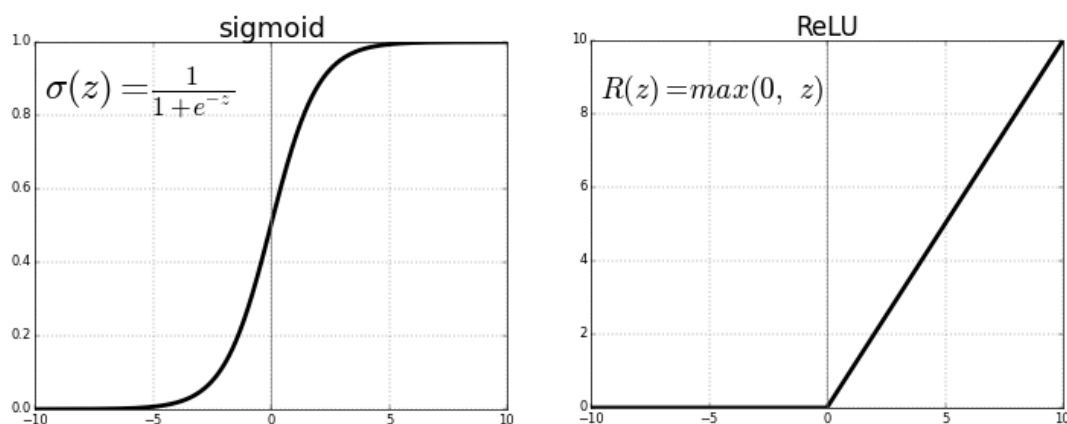


Figura 4. Funcions d'activació Sigmoid i ReLU

Formalment, tenim la funció de sortida d'una neurona,

$$a^{(n+1)} = R(Wa^{(n)} + b)$$

, on,

- a és un vector amb les activacions de la capa n
- W és una matriu amb els pesos de totes les connexions de la primera capa amb la segona capa

- B un vector amb els biaixos de cada una de les neurones de la capa $n + 1$
- R és la funció d'activació ReLU

, tenim que, $a^{(n+1)}$ serà la sortida d'una neurona de la capa $n + 1$.

Un cop s'han calculat totes les activacions a les capes ocultes, la informació arriba a la capa de sortida, que és la responsable de proporcionar la predicció final. El nombre de neurones a la capa de sortida depèn del problema de classificació.

En un problema de **classificació binària** (com per exemple, s'ha detectat un senyal de trànsit o no) normalment tindrem una única neurona a la capa de sortida.

Mentre que en una **classificació multiclasse** (a quina categoria pertany el senyal detectat), tindríem tantes neurones com categories possibles.

En el nostre cas, tractarem un problema de classificació multicapa. En comptes d'usar una funció d'activació com la ReLU, usem la funció **softmax**, que *mapeja* un vector en R a un altre vector en R on la suma dels valors del vector és igual a 1.

D'aquesta manera si tenim un vector R amb les activacions de l'última capa,

$$R = [prob(A^0) + prob(A^1) + \dots + prob(A^n)] = 1$$

, per n neurones o classes a l'última capa, tindrem la probabilitat per cada classe. Per realitzar la predicció, podrem seleccionar la classe amb major probabilitat.

2.5. Xarxes Neuronals Convolucionals

Les xarxes convolucionals (CNN) són xarxes neuronals enfocades al tractament d'imatges. El seu funcionament es basa en reconèixer petits patrons dins d'una imatge per identificar un patró més complex.

Una CNN, a part de tenir una capa d'entrada i una de sortida, ha de tenir mínim dues capes ocultes: una capa convolucional i una capa de *pooling*. La capa de sortida serà una capa totalment connectada.

2.5.1. Capa Convolucional

L'objectiu de la capa convolucional és detectar característiques de la imatge en petites finestres, com arestes, línies, vèrtexs, etc. La unió de diferents capes convolucionals pot detectar característiques amb més detall.

Tindrem tres elements principals en aquesta capa, la imatge d'entrada, un filtre o *kernel*, i la matriu de sortida o mapa de característiques.

Per obtenir el mapa de característiques, usarem el *kernel*, que es mourà pels camps receptius de la imatge per comprovar si hi ha alguna característica present. Aquest procés es denomina **convolució**.

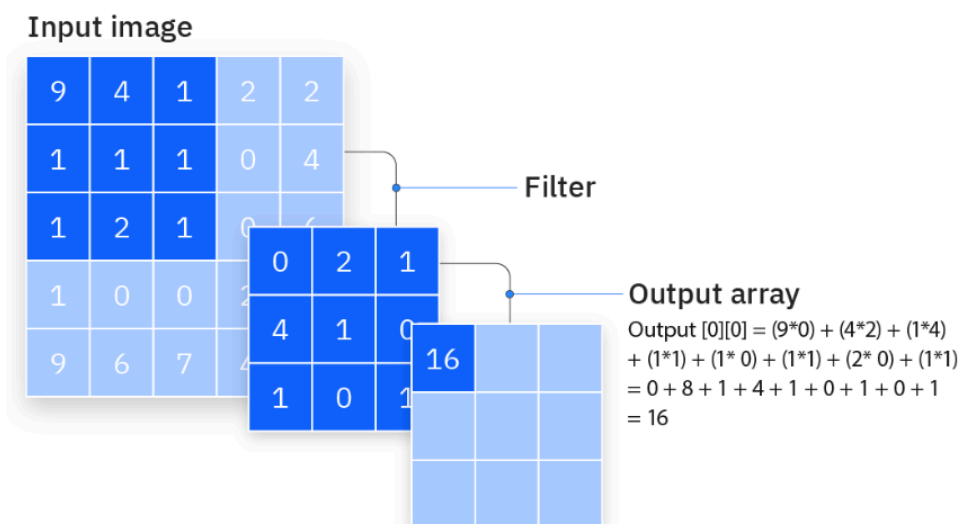


Figura 5. Exemple de càlcul d'un mapa de característiques

El procés és el següent: tenim un filtre, en aquest cas de 3x3, que s'aplica a una àrea 3x3 de la imatge, donant com a resultat un producte escalar entre la imatge i el filtre. Aquest producte escalar s'introdueix a una casella de la matriu de sortida (*figura 5*). El filtre es desplaça d'esquerra a dreta i de dalt a baix fins a recórrer tota la imatge. Aquesta matriu de sortida ja completa serà el **mapa de característiques**.

Observem que per una imatge de 5x5 i un filtre de 3x3 tindrem un mapa de característiques 3x3. Això ens serà útil per simplificar les dades representades i disminuir la capacitat de càlcul.

Normalment, treballarem amb imatges RGB, on la imatge serà representada per una matriu amb una profunditat de tres canals, un per cada component de color (*figura 6*). En aquest cas, el filtre tindria la forma de la figura 7.

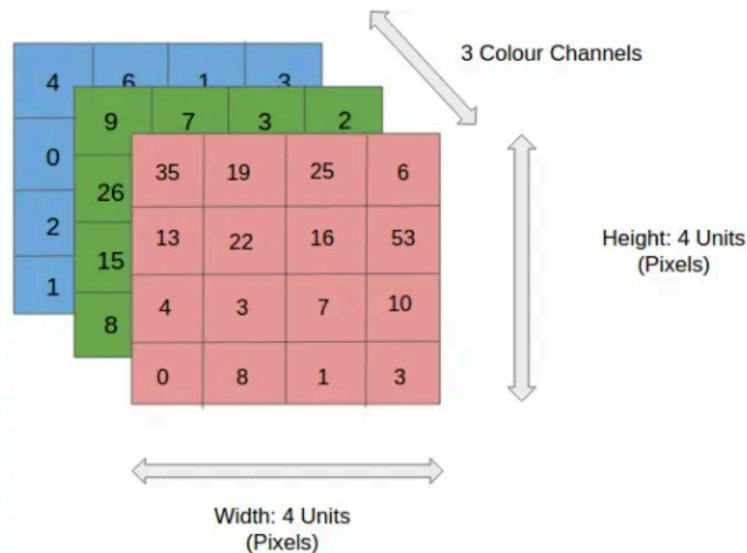


Figura 6. Representació d'una imatge de 3 canals RGB en una matriu

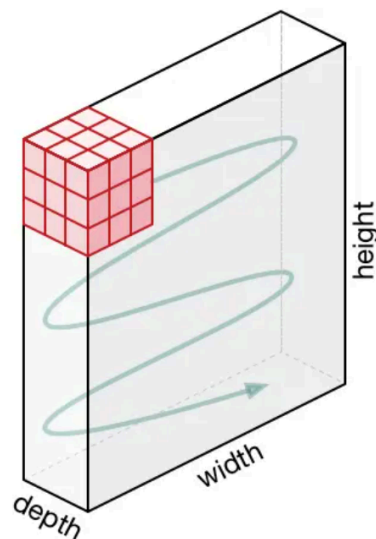


Figura 7. Desplaçament del filtre en una imatge RGB

Després de cada operació de convolució, la CNN aplica una funció d'activació al mapa de característiques, introduint la no-linealitat al model.

La capa de convolució admet una sèrie de paràmetres com el nombre de filtres, la dimensió del filtre o el desplaçament de la finestra (*Stride*).

2.5.2. Capa *Pooling*

Les capes de *pooling* acostumen a anar concatenades de les capes convolucionals, de manera que la sortida de la capa convolucional es converteix en l'entrada de la capa de *pooling*.

L'objectiu principal d'aquesta capa és reduir la mida de la matriu d'entrada. Tot i que això implica una certa pèrdua d'informació, s'utilitza per disminuir la potència computacional necessària pel processament de dades. Cal tenir en compte que les imatges solen tenir dimensions elevades, per la qual cosa és essencial optimitzar-les al màxim.

A més de reduir la mida de la imatge, aquesta capa és útil per extreure característiques que són invariants a rotacions i desplaçaments. Fent que el model sigui més robust a l'hora de trobar característiques amb alguna rotació o canvi d'escala.

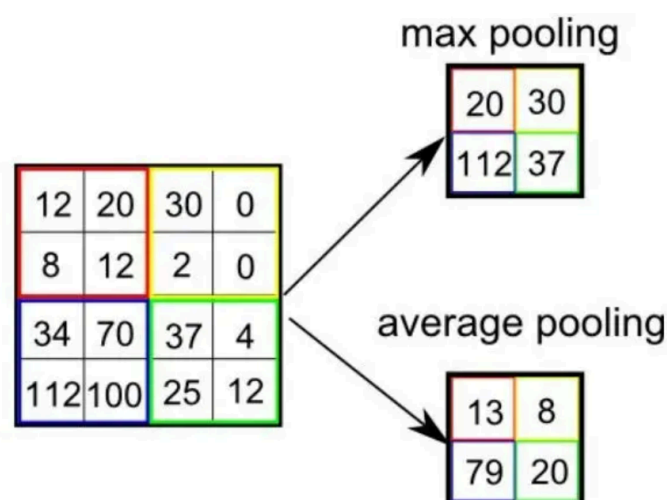


Figura 8. Exemple de max pooling i average pooling

Hi ha dues maneres de fer *pooling* o realitzar l'agrupació per comprimir la imatge (figura 8). El *max pooling* que retorna el valor màxim de la porció d'imatge coberta pel filtre, i l'*average Pooling* que retorna la mitjana de tots els valors de la porció.

La sortida de l'última capa de *pooling* és un conjunt de tensors (matrius) que contenen la informació processada per les capes anteriors.

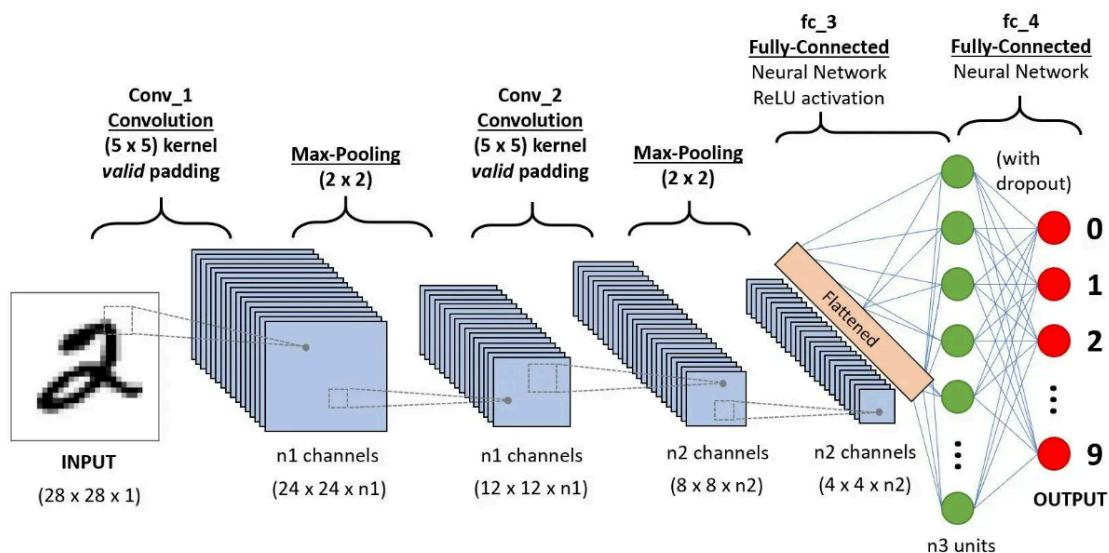


Figura 9. Procés d'una xarxa neuronal convolucional

Els tensors de sortida de la capa de *pooling* són transformats en un vector unidimensional (*flattening*), el que ens permetrà afegir una capa totalment connectada.

2.5.3. Capa totalment connectada

En una NN, una neurona de la capa d'entrada està connectada amb una de la capa de sortida, a través de les connexions de les capes ocultes. Fins ara, en una CNN, els valors dels píxels de la imatge d'entrada no estaven connectats directament amb la capa de sortida.

Per tant, tal com ho fa una NN, hem d'afegir com a mínim, una capa completament connectada, on cada node de la capa de sortida estigui connectada directament amb un node de la capa anterior. Això ens permetrà combinar les característiques processades per extreure patrons globals.

Els pesos d'aquestes connexions són ajustats durant l'entrenament per optimitzar la predicció del model.

Aquesta última capa farà ús de la funció d'activació *softmax* per classificar adequadament la detecció.

3. Models de detecció i classificació d'objectes

3.1. Què és detecció i classificació?

Gran part de la feina en el camp de la visió artificial es basa en el reconeixement d'objectes en imatges o altres tipus de suports multimèdia. Un reconeixement pot abordar dues tasques. La detecció i la identificació o classificació:

Una **detecció** consisteix a localitzar objectes dins d'una imatge, identificant-ne la posició a la imatge mitjançant caixes delimitadores (*bounding boxes*).

Una **classificació** consisteix a assignar etiquetes als objectes detectats en una imatge, identificant la seva categoria o **classe**, entre les definides en el problema.

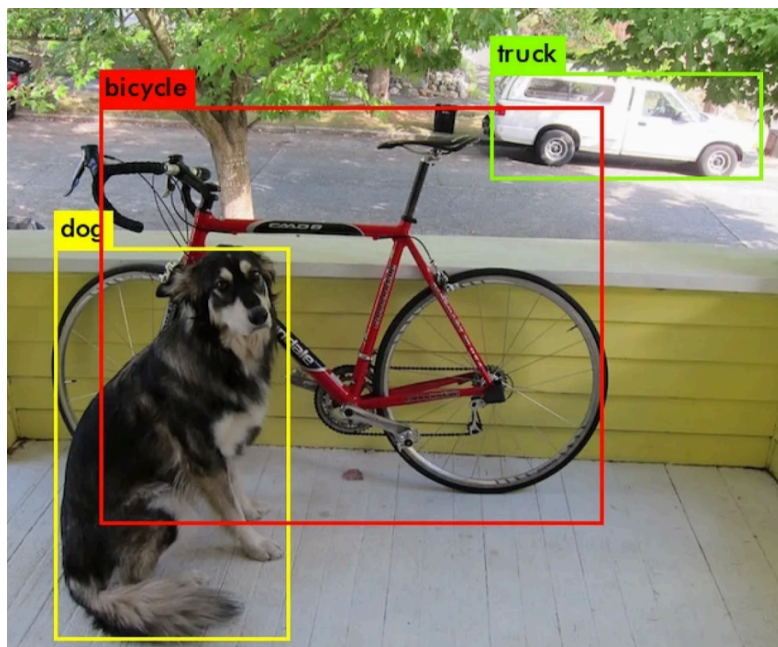


Figura 10. Exemple de detecció i classificació

També trobem el problema de la segmentació, que consisteix a localitzar amb alta precisió el contorn d'un objecte (*figura 11*), però no l'abordarem en aquest projecte.



Figura 11. Exemple de segmentació i classificació

Hi ha versions del model de YOLO que se centren en segmentació. La primera versió i d'altres estan optimitzades només per la detecció.

3.2. YOLO (You Only Look Once)

3.2.1. Introducció

Per solucionar el problema de detecció i segmentació del nostre treball usarem l'algorisme de YOLO.

YOLO, de les inicials *You Only Look Once*, és un algorisme de detecció d'objectes en temps real i amb alta precisió. El seu nom es deu al fet que el sistema analitza tota la imatge una sola vegada. Amb la imatge o el *frame* analitzat, pot realitzar deteccions, segmentacions i classificacions amb un cert grau de confiança.

Una de les principals característiques de YOLO és la seva velocitat sorprenent, ja que la seva versió bàsica processa 45 fps i una versió més lleugera del model aconsegueix arribar fins a 150 fps, superant el processament d'una imatge en temps rècord fins aleshores.

A diferència dels sistemes de detecció tradicionals, que sovint utilitzen classificadors per identificar objectes, YOLO reformula el problema com una tasca de regressió. És a dir, “*dibuixa*” directament les coordenades de l'objecte i la seva classe en una sola passada de la imatge, la qual cosa explica la seva baixa velocitat.

A més, com que YOLO processa tota la imatge alhora, és capaç d'incorporar informació contextual sobre les classes dels objectes. Aquesta capacitat redueix els

errors relacionats amb el fons, que són més freqüents en models com *Fast RCNN*, que analitzen només fragments o parts d'una imatge.

En definitiva, YOLO combina velocitat, precisió i capacitat d'aprenentatge general en un sistema eficient i adequat per a aplicacions en temps real.

3.2.2. Arquitectura

L'arquitectura del YOLO es basa en una xarxa neuronal convolucional de 24 capes convolucionals, seguides de 2 capes totalment connectades. En tenir tantes capes de convolució, les quals ja simplifiquen el mapa de característiques, no cal tenir capes de *pooling*.

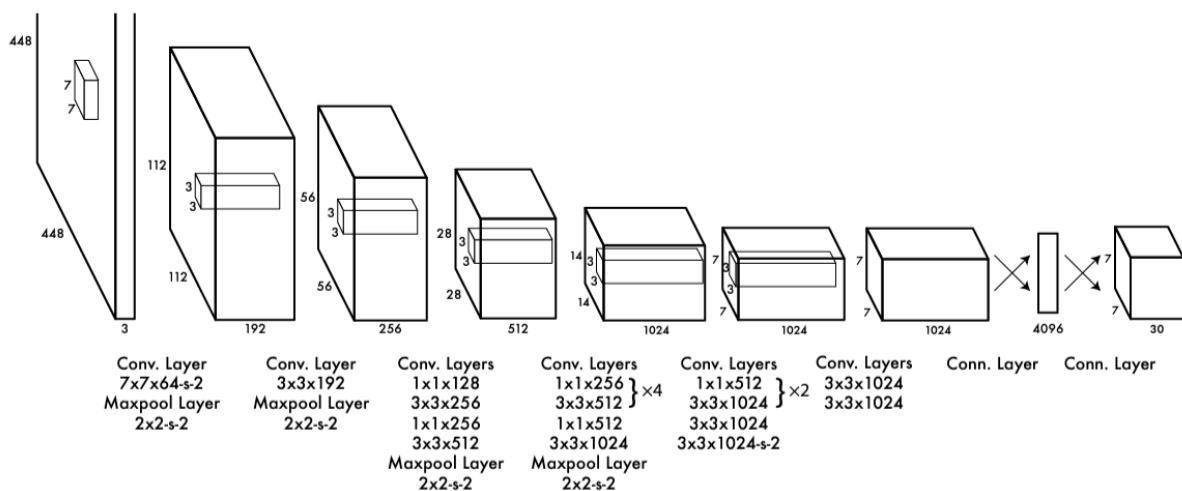


Figura 12. Arquitectura de la CNN de YOLO

3.2.3. Funcionament del model

Pel nostre problema usarem l'aprenentatge supervisat. Passarem al model una imatge i la seva etiqueta corresponent. L'etiqueta ha de contenir la informació sobre les caixes delimitadores i la classe la qual pertany l'objecte.

En el model YOLO, l'etiqueta es guarda en un fitxer de text (.txt), on cada línia correspon a la informació d'una caixa. La informació necessària pel model és la següent:

- Id de la classe
- Coordenada X normalitzada del centre de la caixa delimitadora

- Coordenada Y normalitzada del centre de la caixa delimitadora
- Amplada de la caixa delimitadora
- Alçada de la caixa delimitadora

Mitjançant aquesta informació, el model processarà la informació com un vector de mida 5, més la quantitat total de classes.

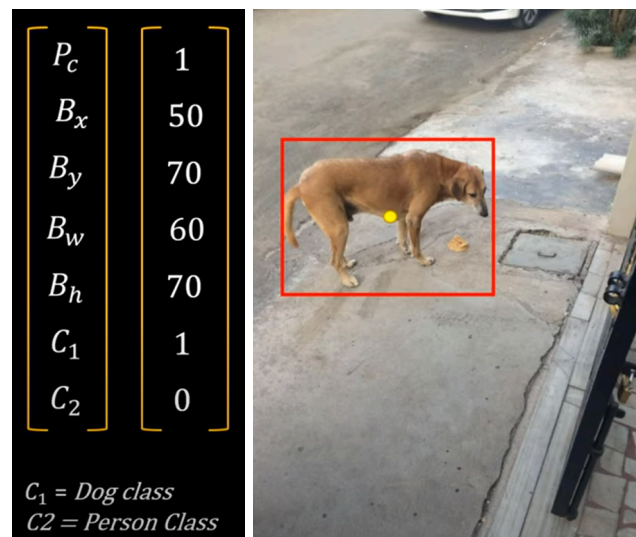


Figura 13. Mostra d'una detecció amb la informació de la bounding box

En l'exemple de la figura 13, es mostra el vector tal com l'entén YOLO. Tindrem la probabilitat de pertànyer l'objecte a una classe gos (P_c), la qual serà 1, evidentment. També tindrem les coordenades del centre (B_x i B_y), i les mides de la caixa delimitadora (B_w i B_h). Finalment, afegim un element per cada classe essent 1 la classe de l'objecte.

Per a imatges sense cap objecte de la classe, tindrem el vector buit (*figura 14*).



Figura 14. Mostra sense cap objecte a detectar

El model sabrà la quantitat de classes diferents que tenim i el seu respectiu nom, mitjançant un fitxer `.yaml` (figura 15).

```
train: /absolute/path/to/dataset/train/images/  
val: /absolute/path/to/dataset/valid/images/  
test: /absolute/path/to/dataset/test/images/  
  
nc: 2  
names: ['dog', 'person']
```

Figura 15. Exemple de fitxer `yaml` per a la figura 13

Un cop tenim un conjunt de dades etiquetades prou extens, podem entrenar la nostra CNN.

Per realitzar una predicció, passarem una imatge com a entrada i la CNN ens hauria de retornar un vector, amb la informació de la caixa delimitadora corresponent a la imatge (figura 16).

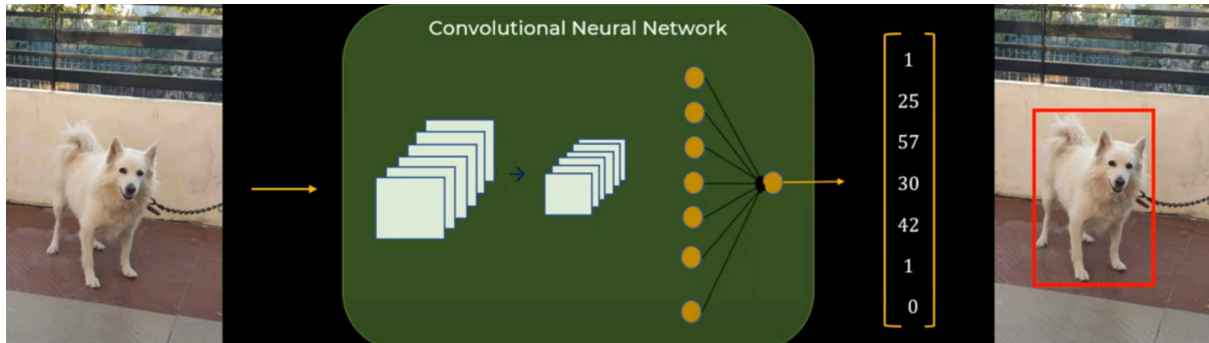


Figura 16. Exemple visual d'una predicció

Ara bé, aquest model funciona correctament per predir un únic objecte a una imatge. Què passaria si tinguéssim una imatge amb 2 gossos? I 10 gossos? En aquests casos determinar la mida del vector de sortida seria complicat i gents eficients. Amb 10 gossos i només 2 classes el vector tindrà una mida de 70 elements. No podríem usar aquest model en temps real.

3.2.4. Detecció multiclasse

Perquè YOLO pugui ser usat en problemes de detecció en temps real, han desenvolupat una manera única i pionera en com el model analitza la imatge.

El model rep una imatge d'entrada, i divideix aquesta imatge en una graella $S \times S$. La cel·la que conté el centre d'un objecte serà l'encarregada de fer la detecció (figura 17).

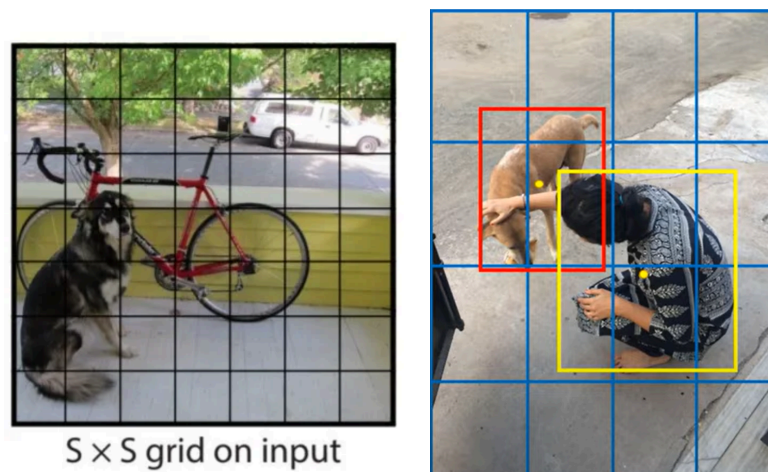


Figura 17. Subdivisió de la imatge en cel·les

Cada una de les cel·les prediu B bounding box amb una confiança associada.

Cada bounding box serà representada per un vector, com els que hem vist anteriorment.

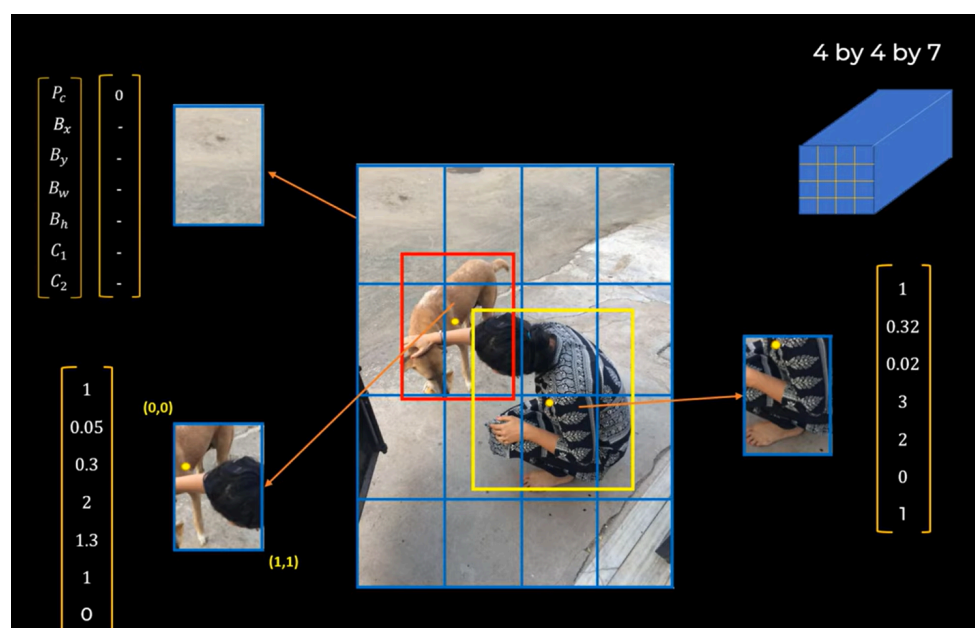


Figura 18. Exemple de detecció multiclasse

Per l'exemple de la figura 18, hem dividit la imatge en 4 x 4 cel·les i per cada una de les cel·les tenim un vector de 7 elements. Per tant, la mida del tensor per la nostra CNN serà de 4 x 4 x 7.

En fase d'entrenament, la CNN processarà la imatge d'entrada i el tensor corresponent.

Un cop el model ha estat entrenat, ja podem obtenir prediccions sobre una imatge. La sortida de la CNN serà de 16 vectors, un vector per cel·la.

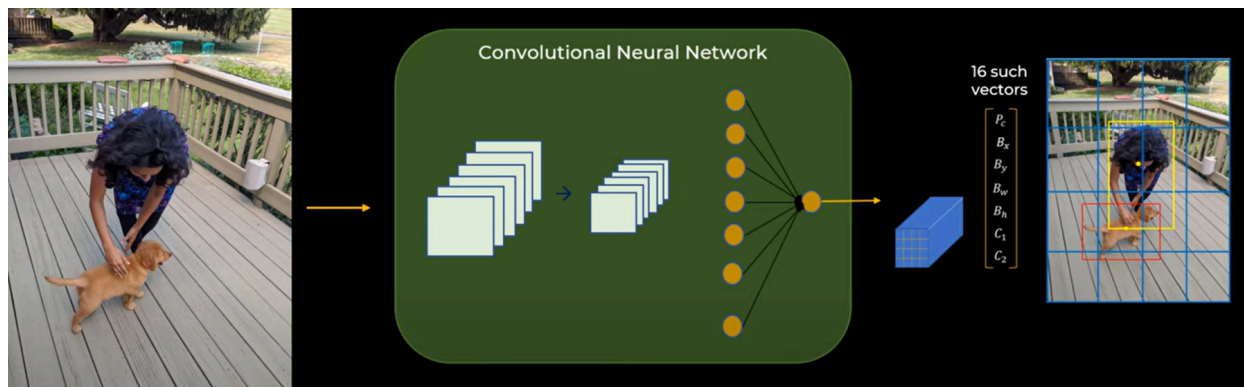


Figura 19. Exemple del procés per realitzar una predicció

Ara bé, el model pot haver detectat múltiples *bounding boxes* pel mateix objecte (figura 20).

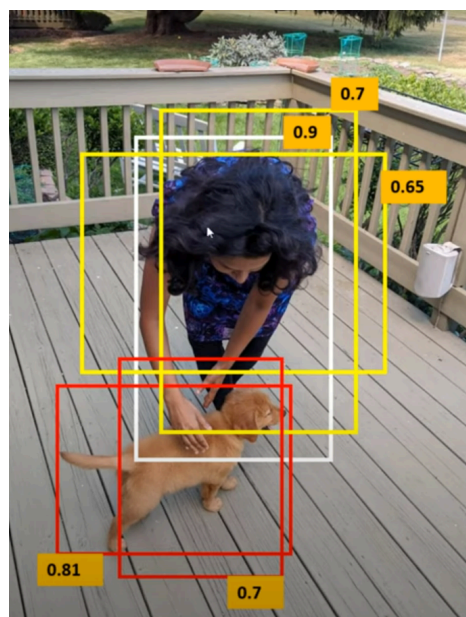


Figura 20. Exemple de detecció amb múltiples caixes delimitadores

Molts pensaran que el més lògic serà quedar-nos amb la *bounding box* amb la confiança més alta d'una classe específica. Però això és incorrecte.

En el cas de la figura 20, podríem tenir altres persones a la imatge. Si ens quedéssim amb la *bounding box* amb major confiança eliminariem la caixa delimitadora de l'altra persona. Per aquest motiu es realitza la tècnica del ***non max suppression***.

Aquesta tècnica es basa en la mètrica *Intersection over Union* (IoU), que avalua la precisió de la *bounding box* per a la detecció d'un objecte. Consisteix a dividir l'àrea que les dues caixes comparteixen, entre l'àrea conjunta d'ambdues caixes. El resultat serà un valor comprès entre 0 i 1.

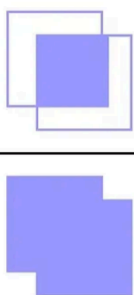
$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$


Figura 21. Càlcul de la *Intersection over Union* (IoU)

El procediment pel *non max suppression* és el següent:

Inicialment, es descarten les *bounding box* amb una confiança baixa, per sota d'un llindar establert. Es calcula el IOU entre cada parella de caixes de la mateixa classe.

El valor d'IoU serà elevat per a les *bounding boxes* que representen el mateix objecte i baix per a aquelles caixes que corresponen a objectes diferents, encara que siguin de la mateixa classe. Per a objectes separats d'una mateixa classe, l'IoU serà proper a 0.

Només les caixes amb un IoU superior al llindar (per exemple, 0.5) són considerades redundants i, per tant, se'n suprimeixen les que tenen puntuacions més baixes. En aquest cas, mantindrem una caixa per cada objecte d'una mateixa classe diferent.

Una altra qüestió és, què passaria si dos o més objectes diferents tenen els seus centres dins de la mateixa cel·la?

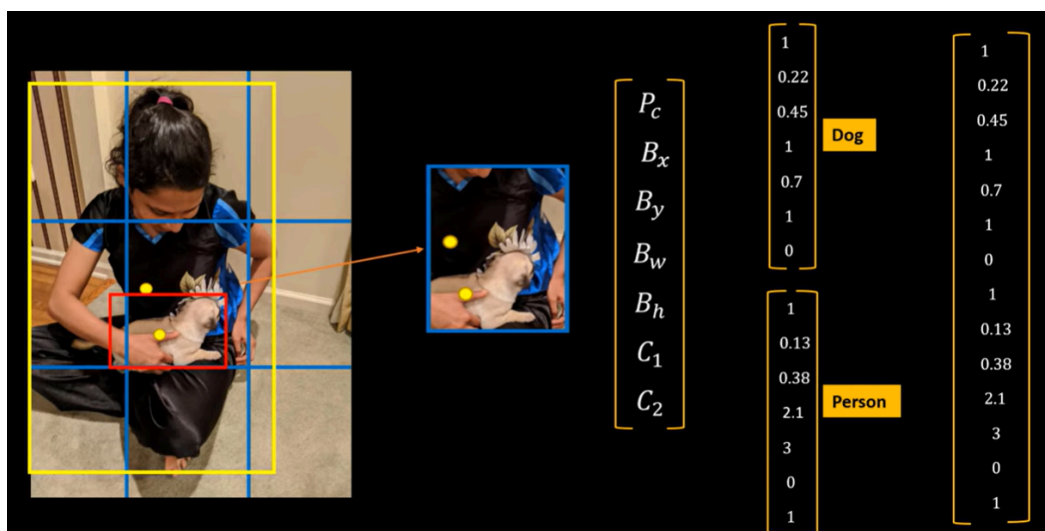


Figura 22. Exemple de detecció multiclasse d'una CNN amb anchor boxes

Una solució no gaire òptima és augmentar la mida del vector N vegades, per N centres en una cel·la (*figura 22*). Aquests vectors es denominen caixes ancorades, i s'utilitzen amb CNN's d' N caixes ancorades. La mida de la CNN augmenta, la qual cosa no és tan eficient com una CNN simple.

Una solució més bona seria augmentar el nombre de cel·les a la imatge. Disminuiria la probabilitat que més d'un objecte caigui dins d'una mateixa cel·la. La mida del tensor seria més petit que utilitzant caixes ancorades, facilitant la feina d'anàlisi de la imatge en una sola passada.

Ara que ja sabem com es realitza una predicció. Com aconseguim la confiança que ve associada?

Aquesta confiança és definida per la probabilitat d'un objecte a la cel·la ($P(Objecte)$) per la IOU entre la caixa predita i la caixa real.

$$Confiança = P(Objecte) \times IOU(Caixa predita, Caixa real)$$

Cada una de les cel·les també prediu les probabilitats de classe per a cada caixa (*figura 23*).

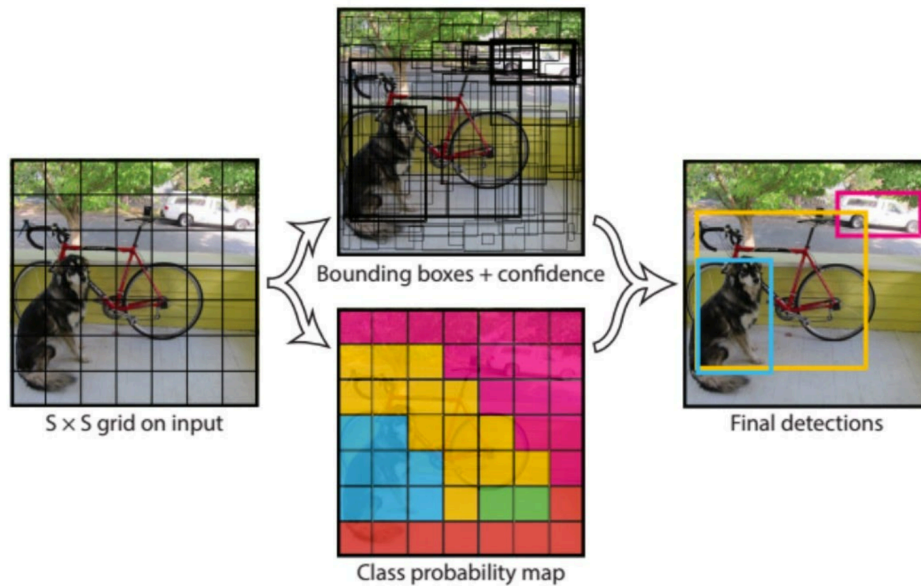


Figura 23. Mapa de probabilitats de classes de la figura 10

La **predicció final** per a cada caixa és la probabilitat combinada entre la confiança de l'objecte i les probabilitats de les diferents classes.

$$\text{Predicció final} = \text{Confiança} \times P(\text{classe}(\text{objecte}))$$

Aquesta predicció final ens determina tant el grau de fiabilitat d'una caixa per un objecte, com la probabilitat de pertànyer l'objecte a una classe.

3.3. Alternatives a YOLO

Avui en dia coneixem una gran quantitat d'algorismes per al reconeixement d'imatges. El millor model el trobarem en funció del problema que vulguem solucionar.

Hi ha algorismes amb una precisió excel·lent, però no es poden utilitzar en aplicacions en temps real a causa de la seva baixa velocitat. D'altres competeixen amb YOLO a nivell de velocitat de predicció, però no ofereixen tanta precisió en el resultat.

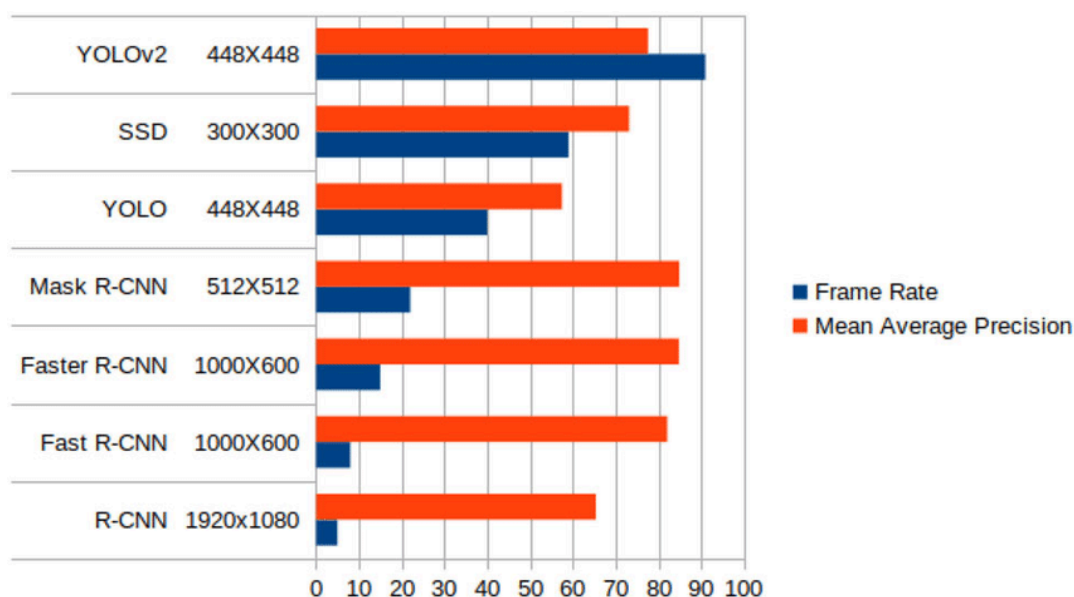


Figura 24. Smart healthcare system-a brain-like computing approach for analyzing the performance of detectron2 and PoseNet models for anomalous action detection in aged people with movement impairments

M'he basat en aquest [article de Jonathan Hui](#) per obtenir una comparativa entre diferents models i arquitectures.

Principalment, parlarem de dos models els quals poden tenir un rendiment similar al YOLO.

El model ***Faster R-CNN*** és una evolució dels models R-CNN i Fast R-CNN, més ràpida en la generació de regions proposades.

El funcionament del model *Faster R-CNN* és el següent: La imatge s'introdueix en una CNN la qual genera un mapa de característiques. Aquest mapa resumeix informació essencial de la imatge conservant-ne detalls importants.

Una altra xarxa anomenada *Region Proposal Network* (RPN), s'encarrega de proposar regions d'interès dins el mapa de característiques. La RPN prediu si l'àrea proposada és el fons de la imatge o conté un objecte, i ajusta les coordenades de les regions per refinar la localització de l'objecte. El resultat són un conjunt de regions suggerides les quals poden contenir objectes.

El que fa que el Faster R-CNN sigui un model realment innovador és la seva tècnica de *Region of Interest (ROI) pooling*. Aquesta característica ajuda el model a classificar imatges dividint la regió d'interès de la imatge entrant, en blocs més petits.

La detecció es realitza en dues etapes, on, en la primera etapa, es busquen les regions d'interès, i en la segona fase, filtra i classifica les propostes usant el ROI. Tot i que YOLO encara és més ràpid que el Faster R-CNN (a causa del procés de dues passades a la imatge del Faster R-CNN), aquest últim es considera més precís que YOLO en les seves primeres versions (*figura 25*).

Method	mAP	FPS	batch size	# Boxes	Input resolution
Faster R-CNN (VGG16)	73.2	7	1	~ 6000	~ 1000 × 600
Fast YOLO	52.7	155	1	98	448 × 448
YOLO (VGG16)	66.4	21	1	98	448 × 448
SSD300	74.3	46	1	8732	300 × 300
SSD512	76.8	19	1	24564	512 × 512
SSD300	74.3	59	8	8732	300 × 300
SSD512	76.8	22	8	24564	512 × 512

Speed is measure with a batch size of 1 or 8 during inference.

Figura 25. Comparació de diferents models validats pel conjunt Pascal VOC 2007 test.

Per altra banda, el model **SSD (Single Shot Multibox Detector)** utilitza també un model de *deep learning* per detectar objectes en temps real. És una de les alternatives a YOLO més potents. Destaca per una alta precisió tot i oferir-la en alta velocitat. És eficient ja que només utilitzen una sola xarxa neuronal per predir ubicacions precises dels objectes.

Mentre que YOLO divideix la imatge en una graella i fa prediccions per a cada cel·la, el model SSD usa múltiples mapes de característiques de diferents resolucions per predir la posició dels objectes, cosa que millora la detecció en diferents escales i augmentant la flexibilitat del model.

En general, SDD i YOLO ofereixen un bon rendiment per a la detecció d'objectes grans, però no és tan precís per a la detecció d'objectes petits.

Detection Frameworks	Train	mAP	FPS
Fast R-CNN [5]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[15]	2007+2012	73.2	7
Faster R-CNN ResNet[6]	2007+2012	76.4	5
YOLO [14]	2007+2012	63.4	45
SSD300 [11]	2007+2012	74.3	46
SSD500 [11]	2007+2012	76.8	19
YOLOv2 288 × 288	2007+2012	69.0	91
YOLOv2 352 × 352	2007+2012	73.7	81
YOLOv2 416 × 416	2007+2012	76.8	67
YOLOv2 480 × 480	2007+2012	77.8	59
YOLOv2 544 × 544	2007+2012	78.6	40

Figura 26. Comparació de diferents models validats en el conjunt Pascal VOC 2007 test.

A la figura 25, podem observar com en la primera versió de YOLO, tant amb la seva xarxa completa, com en la simplificada capaç de processar 155 fps, no supera en precisió els models SSD i *Faster R-CNN*.

A la figura 24, es considera el model *Faster R-CNN* com el més precís fins aleshores, però la seva velocitat no li permet competir en aplicacions de temps real com sí que ho fa el SSD.

A la figura 26, ja amb una versió millorada del YOLO (YOLOv2), obtenim una precisió extraordinària en imatges d'alta resolució. Per imatges amb una resolució menor, la velocitat de predicció augmenta, però la precisió disminueix (cosa que iguala els resultats del SSD).

Cada vegada els models van millorant, se'n desenvolupen noves versions, i es fan prediccions amb conjunts de dades diferents. Per tant, és complicat considerar el model amb millor rendiment.

Les dades i figures anteriors les hem obtingut dels *papers* originals dels models mencionats.

4. Entrenament del model i *fine-tuning*

En aquesta secció repassarem com es realitza un entrenament i explorarem la tècnica del *fine-tuning*, utilitzada en el treball.

Ja sabem com està format un model a nivell de la xarxa neuronal, per tant, ara només ens queda saber com podem entrenar-lo per aconseguir un sistema fiable i amb bon rendiment.

El nostre model aprendrà d'un conjunt de dades les quals estan formades per imatges i les respectives etiquetes que formen les caixes delimitadores de cada detecció.

Per aconseguir el millor rendiment del nostre sistema, necessitem tenir la CNN amb els pesos òptims. Com podem obtenir aquests pesos mitjançant les mostres etiquetades?

El procés d'entrenament d'un model consisteix en el següent:

Començarem establint els pesos del nostre model aleatòriament. Mitjançant les dades, realitzarem un *Forward Propagation*, que bàsicament és passar-li una imatge al model perquè realitzi una predicció sobre ella. Mitjançant una funció de pèrdua, compararem la predicció generada amb l'etiqueta real i obtindrem un error.

En el cas de YOLO, la **funció de pèrdua** utilitzada combina diversos components de penalització per la detecció i classificació d'objectes. La funció de pèrdua es defineix com

$$L_{total} = L_{coord} + L_{conf} + L_{class}$$

, on,

- L_{coord} és la pèrdua de coordenades. Penalitza la diferència entre les coordenades predites i les reals de les bounding boxes.
- L_{conf} és la pèrdua de confiança. Penalitza la diferència entre la confiança de la detecció predita i la real (que és 1).
- L_{class} és la pèrdua de classificació. Penalitza la diferència entre les probabilitats de classe predites i les reals.

Realitzem el procés de *Backward Propagation* que consisteix a calcular com canviaria la pèrdua global respecte petits canvis en els pesos. És a dir, calculem el gradient de la funció de pèrdua respecte a cada pes de la xarxa i utilitzem un algoritme d'**optimització**, com el *gradient descent* (figura 27), per ajustar els pesos en la direcció del gradient negatiu, de manera que minimitzem la pèrdua global.

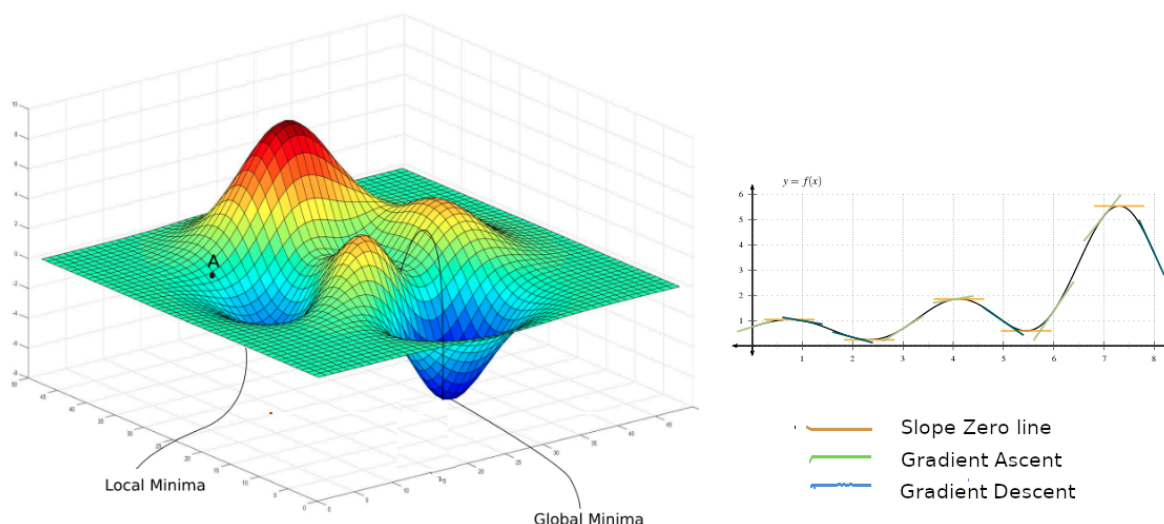


Figura 27. Visualització en 2D i 3D del descens del gradient

El model actualitzarà els pesos cada n mostres. Aquest valor n es coneix com el **batch size**. Totes les mostres es processaran en paral·lel, per la qual cosa hem de tenir un valor de *batch size* adient en funció dels recursos de la nostra memòria RAM o VRAM en el cas de les GPUs.

Aquest procés es repeteix durant diverses iteracions, conegudes com a **èpoques**. Haurem completat una època quan hàgim passat per tot el conjunt. A mesura que s'executen més èpoques, el model ajusta els seus pesos per reduir la pèrdua global i aconseguir una millor capacitat de predicció. L'objectiu és fer les suficients èpoques fins que l'error global convergeix.

Per entrenar els nostres models utilitzarem la tècnica del **fine-tuning**, que consisteix a aprofitar un model ja entrenat i adaptar-lo (entrenar-lo de nou) per a una tasca específica.

En lloc d'entrenar el model des de zero, un procés que requereix grans quantitats de dades i recursos computacionals, el *fine-tuning* ajusta lleugerament els pesos del model preentrenat per optimitzar-lo pel problema concret.

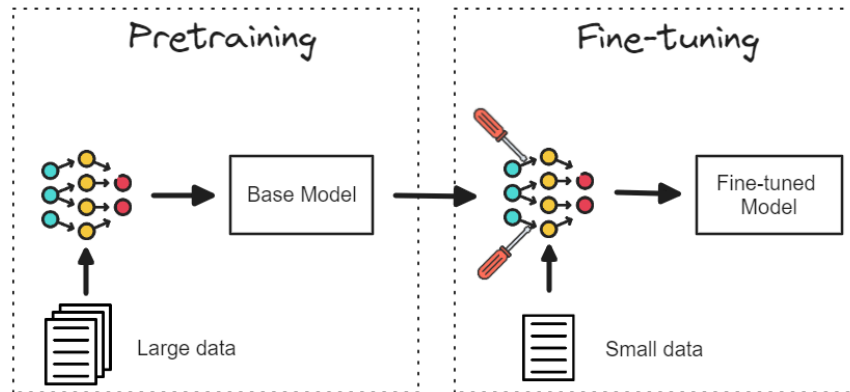


Figura 28. Representació de la tècnica de *fine-tuning*

En l'àmbit de la CNN, trobem que les primeres capes convolucionals, que són les encarregades de detectar característiques bàsiques com contorns i textures, es mantenen congelades perquè ja són útils i no cal reajustar-les. Les capes més profundes seran les encarregades de realitzar l'ajustament.

Pel mateix procés de *backpropagation*, s'ajusten els pesos de les últimes capes de la xarxa. La capa de sortida també s'ajusta per adaptar-se al nombre i tipus de classes, del nou conjunt d'entrenament.

5. Llibreries i recursos computacionals

Per resoldre el nostre problema de detecció i classificació de senyals, hem utilitzat una sèrie de recursos que ens han facilitat el procés. A continuació ús els detallo.

5.1. *Ultralytics*

Ultralytics és una empresa que ha desenvolupat millores en els models de YOLO ja existents. És de codi obert, i destaca per proporcionar una API fàcil de fer servir, així com eines per entrenar models personalitzats, i mètriques per avaluar els models. En el meu cas, l'he usat per entrenar, validar i testejar els models obtinguts.

5.2. *Google Colab*

Google Colab és un entorn de programació de Google, el qual ens permet executar codi *Python* fent ús dels recursos computacionals del núvol, com la GPU i la TPU.

Aquesta eina ens permet crear un fitxer *notebook* i executar el codi sense haver de tenir un entorn configurat. A l'hora de treballar amb dades, el *notebook* et permet executar el codi d'una sola cel·la, així doncs pots executar un programa pas a pas com et convingui.

Google ofereix un servei il·limitat per a l'ús de la CPU. No obstant això, l'ús de la GPU i la TPU està restringit a un temps determinat. Tot i no haver trobat informació exacta sobre el límit diari, després de realitzar desenes d'entrenaments amb diferents nombres d'èpoques, he observat que aquestes limitacions depenen del nombre d'entrenaments realitzats i la durada de cadascun d'ells. Per tant, en ocasions quan he hagut d'ajustar varies vegades els paràmetres de l'entrenament, m'he quedat sense temps per realitzar els nous entrenaments amb la GPU o TPU.

Havia sentit a parlar de les TPUs però ni molt menys coneixia el seu funcionament. Bàsicament, les Unitats de Processament Tensorial, a diferència de les GPUs, estan dissenyades específicament per executar de forma optimitzada, operacions de matrius, utilitzades en les xarxes neuronals.

A la secció 9.1 explico quina unitat de processament he acabat fent servir.

6. Conjunts de dades

6.1. Introducció

Les dades que utilitzarem per entrenar la CNN són l'única font de coneixement del model, per tant, és important tenir un conjunt de dades complet i divers en multitud de contextos.

Pel problema de detecció i classificació de senyals de trànsit, el conjunt de dades ha de contenir una gran quantitat d'imatges de senyals de trànsit. És important incloure mostres de senyals en diferents condicions meteorològiques, de diferents mides, qualitats i angles de visió. També cal un bon balanceig entre classes de senyals per evitar qualsevol mena de biaix, i cal tenir imatges tant recents com antigues per reflectir l'evolució dels senyals amb el temps.

6.2. EuRoSign

El primer conjunt de dades l'anomenem EuRoSign (*European Road Signs*). És un conjunt d'imatges de senyals de trànsit **europesos**. Aquest conjunt destaca per tenir imatges en condicions de llum extremes. Podem observar com els senyals es visualitzen amb poca claredat (*figura 29*).

00562.jpg
1 0.5275735294117647 0.5425 0.03602941176470588 0.0525
0 0.4176470588235294 0.6025 0.016176470588235296 0.0275



00410.jpg
1 0.611764705882353 0.705 0.022058823529411766 0.035
0 0.6125 0.735625 0.014705882352941176 0.02375
1 0.3573529411764706 0.705625 0.023529411764705882 0.03375
0 0.35808823529411765 0.73375 0.014705882352941176 0.0225



00164.jpg
0 0.2797794117647059 0.571875 0.016911764705882352 0.02875
0 0.54375 0.555 0.015441176470588236 0.0275



00572.jpg
3 0.5294117647058824 0.53 0.03235294117647059 0.0525



00026.jpg
0 0.3474264705882353 0.53875 0.028676470588235293 0.05



00337.jpg
0 0.8180147058823529 0.6775 0.025735294117647058 0.045
0 0.35404411764705884 0.683125 0.027205882352941177 0.04625



Figura 29. Mostres del conjunt EuRoSign

Per treballar amb el problema de classificació hem buscat un *dataset* el qual classifiqui els senyals en prohibit (*prohibitory*, 0), perill (*danger*, 1), obligatori (*mandatory*, 2) i altres (*other*, 3).



0 0.7040441176470589 0.4775 0.04632352941176471 0.08

Figura 30. Mostra d'imatge amb etiqueta

A la figura 30 podem observar una imatge amb un senyal que pertany a la classe 0, *prohibitory*.

A taula 2 i a la figura 31 es pot veure un resum de la informació del conjunt EuRoSign.

Conjunt:	EuRoSign
Quantitat d'imatges/etiquetes totals	741

Pes total del conjunt	250.66 MB
Número de classes totals	4
Classe 0	<i>Prohibitory</i>
Classe 1	<i>Danger</i>
Classe 2	<i>Mandatory</i>
Classe 3	<i>Other</i>
Mostra d'una imatge N (00894.jpg)	
Format de les etiquetes	<class_id> <x_center> <y_center> <width> <height>
Mostra de l'etiqueta de la imatge N (00894.txt)	<pre> 0 0.2963235294117647 0.67625 0.020588235294117647 0.035 0 0.8180147058823529 0.685625 0.028676470588235293 0.04875 3 0.8158088235294118 0.628125 0.03602941176470588 0.06125 </pre>
Contextualització de l'etiqueta d'N	Trobem 2 senyals de la classe 0 (prohibició) i 1 senyal de la classe 3 (altres)
Senyals trobats al conjunt i no considerades	- Semàfors (tot i no ser considerats senyals) - Senyals d'informació rectangulars blaves (p.e. pàrquing)

Taula 2: Resum del conjunt EuRoSign.

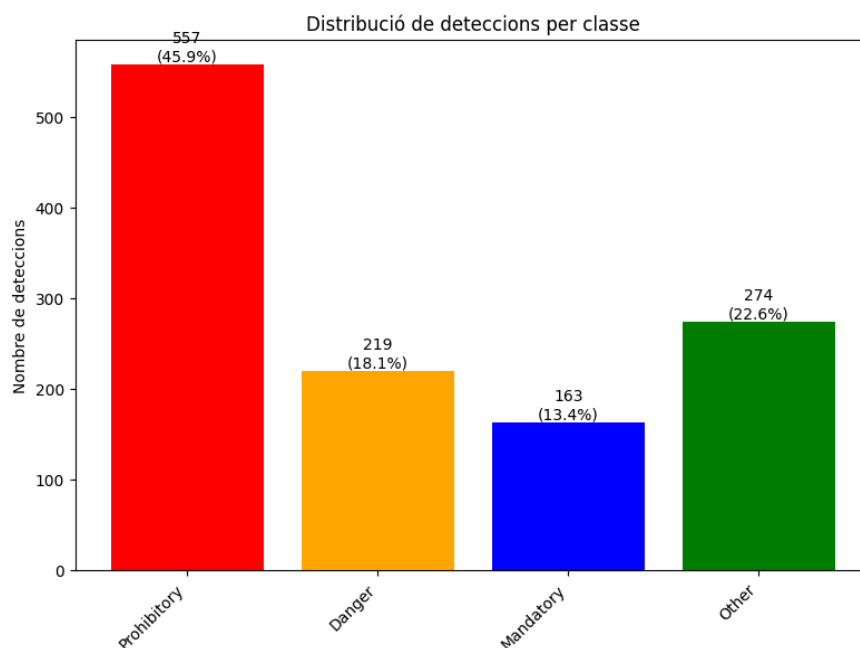


Figura 31. Distribució de les classes del conjunt EuRoSign

Si ens fixem en la figura 31, podem veure com les classes del conjunt EuRoSign estan desbalancejades. Pràcticament, tenim el doble de mostres de senyals de prohibició que de les altres. Aquest desequilibri pot afectar el rendiment del model, provocant una millor capacitat de detecció per a la classe *prohibitory* que de les altres.

6.3. GTSDDB - German Traffic Sign Detection Benchmark

Aquest conjunt de dades està format per imatges de senyals de trànsit d'**Alemanya**, i està enfocat per a la detecció de senyals en imatges. És àmpliament utilitzat en investigació i desenvolupament de sistemes de visió artificial per a vehicles autònoms, i assistència a la conducció (ADAS).

El *dataset* té 900 imatges classificades en 43 classes. D'aquestes 43 classes es diferencien 4 subconjunts (*prohibitory*, *danger*, *mandatory*, *other*):

- 0 = speed limit 20 (prohibitory)
- 1 = speed limit 30 (prohibitory)
- 2 = speed limit 50 (prohibitory)
- 3 = speed limit 60 (prohibitory)
- 4 = speed limit 70 (prohibitory)

5	= speed limit 80 (prohibitory)
6	= restriction ends 80 (other)
7	= speed limit 100 (prohibitory)
8	= speed limit 120 (prohibitory)
9	= no overtaking (prohibitory)
10	= no overtaking (trucks) (prohibitory)
11	= priority at next intersection (danger)
12	= priority road (other)
13	= give way (other)
14	= stop (other)
15	= no traffic both ways (prohibitory)
16	= no trucks (prohibitory)
17	= no entry (other)
18	= danger (danger)
19	= bend left (danger)
20	= bend right (danger)
21	= bend (danger)
22	= uneven road (danger)
23	= slippery road (danger)
24	= road narrows (danger)
25	= construction (danger)
26	= traffic signal (danger)
27	= pedestrian crossing (danger)
28	= school crossing (danger)
29	= cycles crossing (danger)
30	= snow (danger)
31	= animals (danger)
32	= restriction ends (other)
33	= go right (mandatory)
34	= go left (mandatory)
35	= go straight (mandatory)
36	= go right or straight (mandatory)
37	= go left or straight (mandatory)
38	= keep right (mandatory)
39	= keep left (mandatory)
40	= roundabout (mandatory)
41	= restriction ends (overtaking) (other)
42	= restriction ends (overtaking (trucks)) (other)

Taula 3. Classificació dels tipus de senyals del conjunt GTSDb per classes

Per equiparar els conjunts a l'hora d'avaluar els models, he decidit treballar només amb 4 classes, les mateixes que en el conjunt EuRoSign (*prohibitory*, *danger*, *mandatory*, *other*). A la taula 3 podem veure el subconjunt que pertany cada classe entre parèntesis pel nostre cas.



Figura 32. Mostres del conjunt GTSDb

A la taula 4 es pot veure un resum del conjunt GTSDb.

Conjunt:	GTSDb
Quantitat d'imatges/etiquetes totals	900
Pes total del conjunt	301.79 MB
Nombre de classes totals	4 (per aplicacions més complexes es pot fer ús de les 43 classes)
Classe 0	<i>Prohibitory</i>

Classe 1	<i>Danger</i>
Classe 2	<i>Mandatory</i>
Classe 3	<i>Other</i>
Mostra d'una imatge N (00806.jpg)	
Format de les etiquetes	<class_id> <x_center> <y_center> <width> <height>
Mostra de l'etiqueta de la imatge N (00806.txt)	<pre> 0 0.681250 0.526250 0.038971 0.067500 2 0.280147 0.566250 0.020588 0.035000 </pre>
Contextualització de l'etiqueta d'N	Trobem un senyal de la classe 0 (prohibició) i un senyal de la classe 2 (obligació)

Taula 4: Resum del conjunt GTSDb.

6.4. Conjunt COCO

El conjunt de dades COCO (*Common Objects in Context*) és un dels *datasets* més utilitzats en la visió per computador. Conté una gran varietat d'imatges etiquetades amb objectes en diversos contextos reals:

- Més de 330.000 imatges, de les quals 200.000 estan etiquetades.
- Inclou un total d'aproximadament 1,5 milions d'objectes etiquetats.
- Conté 80 classes d'objectes diferents (des de persones, animals i vehicles fins a objectes quotidians com ampolles, cadires i plantes.)
- Les imatges no només mostren els objectes, sinó que també inclouen els seus entorns naturals, com espais públics, habitatges, carrers, etc.

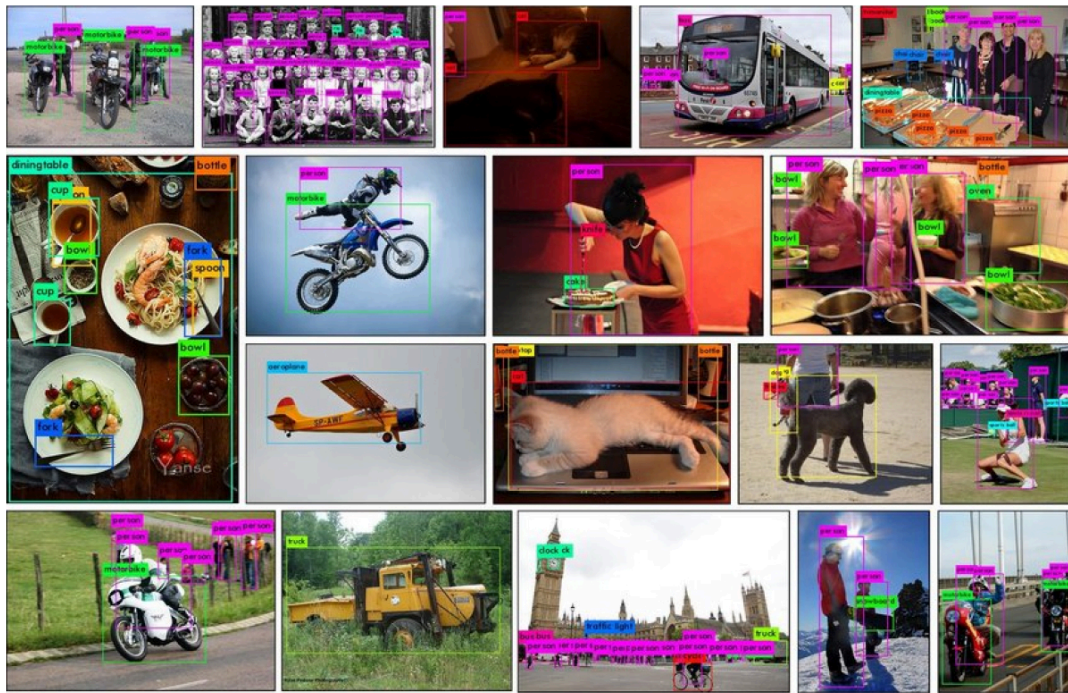


Figura 33. Exemple de deteccions sobre el conjunt COCO

Aquest conjunt de dades sovint s'utilitza per entrenar un model des de zero. Els objectes quotidians del conjunt en infinitat de contextos, fan de COCO un conjunt excel·lent per realitzar el primer entrenament i tenir un model ben robust. A partir d'aquest model preentrenat, es poden realitzar nous entrenaments amb conjunts de dades específics adaptats al problema concret de detecció i classificació, mitjançant tècniques de *fine-tuning*.

6.5. Preparació dels conjunts

A causa del problema esmentat al punt 9.1, he decidit programar a l'entorn de Google Colaboratory tal com explico al punt 5.2. Al programar des del núvol, he preferit tenir emmagatzemades les dades a Google Drive. Això m'ha permès tenir una ràpida i fàcil manipulació dels conjunts de dades.

Hem preparat tres *notebooks* (fitxers compatibles amb *google colab*) pel treball. Un per preparar les dades del conjunt EuRoSign (annex 11.1), el segon per preparar les de GTSDb (annex 11.2), i el tercer per realitzar tot el procés d'entrenament i validació dels models (annex 11.3).

El conjunt EuRoSign ja tenia el format correcte de YOLO, així que simplement hem realitzat el procés de separar les dades per a l'entrenament i la validació.

En el cas del conjunt GTSDb, les imatges estaven en una carpeta en format .ppm. Les etiquetes estaven en un fitxer .txt amb el següent format:

`<img_id.ppm> <leftCol> <topRow> <rightCol> <bottomRow> <ClassID>`

Per preparar els conjunts de dades amb el format correcte, he programat els *scripts* ja mencionats.

Un exemple dels canvis que hem hagut de fer ha sigut calcular les coordenades del centre de la *bounding box*, per tal de formar correctament l'etiqueta.

Per realitzar els entrenaments i les validacions dels models hem dividit ambdós conjunts en *train* (entrenament) i *val* (validació), usant la llibreria de *python* *sklearn*.

El conjunt d'entrenament ens servirà per entrenar el model. Les dades del conjunt de validació les utilitzarem per avaluar el model. Aquest últim conjunt s'usarà per comprovar com de bé generalitza el model en dades mai vistes. Ens proporcionarà les mètriques finals per a comparar i avaluar els diferents models.

La proporció entre conjunts serà del 80% pel conjunt d'entrenament i 20% pel de validació. Normalment en problemes de *deep learning* s'usen proporcions similars a aquestes.

Quan l'objectiu és millorar recurrentment el model, el que es fa és tenir un altre conjunt anomenat *test*, el qual servirà per fer una predicció final amb dades mai vistes. El subconjunt de *validació* passaria a ser usat per avaluar el model per a cada entrenament realitzat.

Perquè el nostre model sàpiga de quin subconjunt ha d'agafar les dades i a quina classe pertany cada detecció, s'utilitza un fitxer de configuració *.yaml* (figures 3 i 4).

```
# Creem un fitxer de configuració YAML per al dataset EuRoSign
%%writefile /content/EuRoSign.yaml

train: /content/EuRoSign/train/
val: /content/EuRoSign/val/

nc: 4
names: ['prohibitory', 'danger', 'mandatory', 'other']
```

Figura 34. Fitxer *.yaml* per la configuració d'EuRoSign.

```
# Creem un fitxer de configuració YAML per al dataset GTSDb
%%writefile /content/GTSDb.yaml

train: /content/GTSDb/train/
val: /content/GTSDb/val/

nc: 4
names: ['prohibitory', 'danger', 'mandatory', 'other']
```

Figura 35. Fitxer .yaml per la configuració de GTSDb.

7. Mètriques d'avaluació

Abans d'avaluar un model, és essencial conèixer les mètriques que permeten analitzar el seu rendiment. Un cop clarificats aquests conceptes, identificarem les més adequades per a l'avaluació dels models.

Per un problema de detecció i en específic pel nostre cas considerem que un *true positive* (TP) es dona quan el model detecta un senyal present a la imatge. I un *true negative* (TN) el cas quan no es fa la detecció perquè no hi ha senyal.

Els casos erronis són el *false positive* (FP) que ocorre quan el model detecta un objecte que no és senyal, i els *false negative* (FN) quan no detecta un senyal que sí que està present a la imatge.

- **Precision**

La precisió és la proporció de deteccions correctes (TP) respecte a totes les deteccions realitzades (TP + FP). Aquesta mètrica quantifica la precisió del model a l'hora de realitzar prediccions positives. És a dir,

$$Precision = TP / (TP + FP)$$

La precisió és crucial quan el cost dels *false positives* és elevat, o quan l'objectiu és minimitzar les deteccions errònies. Per exemple, en un model de conducció autònoma, és fonamental tenir una precisió alta per evitar falses alarmes, com interpretar un objecte inofensiu com un obstacle. Això ajuda a garantir que el sistema només alerti quan realment hi ha un perill, reduint així la probabilitat de decisions incorrectes o accions innecessàries.

- **Recall** (Sensitivity or **True Positive Rate**)

El *recall* mesura la proporció de deteccions correctes (TP) respecte al nombre total d'objectes a detectar (TP + FN). Aquesta mètrica mesura la capacitat del model per detectar correctament totes les instàncies reals d'un objecte en un conjunt de dades.

$$Recall = TP / (TP + FN)$$

En situacions en què és crític no perdre cap detecció (com en videovigilància o detecció d'anomalies mèdiques), un *recall* elevat és important, ja que es preferible detectar erròniament més objectes (falsos positius) que passar per alt un cas positiu real (falsos negatius). En aquests contextos, el *recall* és clau per garantir que els casos importants no es perdin.

- **1 - Specificity (o False Positive Rate)**

En alguns casos s'utilitza aquesta mètrica la qual és la proporció de deteccions d'objectes que no hi són (FP) respecte als objectes totals que no hi són (FP + TN).

$$FPR = FP / (FP + TN)$$

És útil on el cost de tenir falses alarmes és elevat. En el cas de la detecció en proves mèdiques, haver de fer tractaments innecessaris.

- **F1 Score**

Hi ha aplicacions en què és important trobar un equilibri entre precisió i *recall*, com en detecció de correu brossa o en classificació de comentaris ofensius, on tant els *false positives* com els *false negatives* poden afectar l'experiència de l'usuari. En aquests casos, sovint s'utilitza la mètrica *F1 Score*, que és la **mitjana harmònica entre precisió i recall**, per obtenir una mesura global de rendiment que tingui en compte els dos factors.

$$F1\ Score = 2 \cdot Precision \cdot Recall / (Precision + Recall)$$

En aplicacions on tant els *false positives* (deteccions incorrectes) com els *false negatives* (objectes no detectats) tenen conseqüències greus, l'*F1 Score* és útil per trobar aquest equilibri. La **mitjana harmònica** penalitza els casos en què un valor és molt baix, per tant, l'*F1 Score* només serà alt si tant la precisió com el *recall* són alts.

- **Accuracy**

L'*accuracy* mesura el percentatge de prediccions correctes (no només deteccions) que un model fa sobre el total de prediccions realitzades.

$$Accuracy = \text{Correct Predictions} / \text{Total Predictions}$$

I explícitament,

$$Accuracy = TP + TN / TP + TN + FP + FN$$

L'*accuracy* és una mètrica útil en situacions amb **classes equilibrades** (on el nombre de casos positius i negatius és similar). No obstant això, si hi ha un fort **desequilibri de classes**, pot donar una visió distorsionada del rendiment del model.

Per exemple, en una tasca on el 95% de les mostres són de la classe negativa, un model que sempre predigui "negatiu" tindrà una *accuracy* del 95%. Ara bé, això no reflecteix un bon rendiment amb els casos positius.

En aquestes situacions de desequilibri de classes, és preferible utilitzar altres mètriques, com el **recall**, la **precisió** o l'**F1 Score**, per obtenir una visió més acurada del rendiment del model.

- **Intersection over Union (IoU)**

És una mesura usada per avaluar la precisió de les prediccions en tasques de **detecció**. IoU mesura el **grau de superposició** entre la regió predita pel model i la regió real (o *ground truth*) de l'objecte en una imatge.

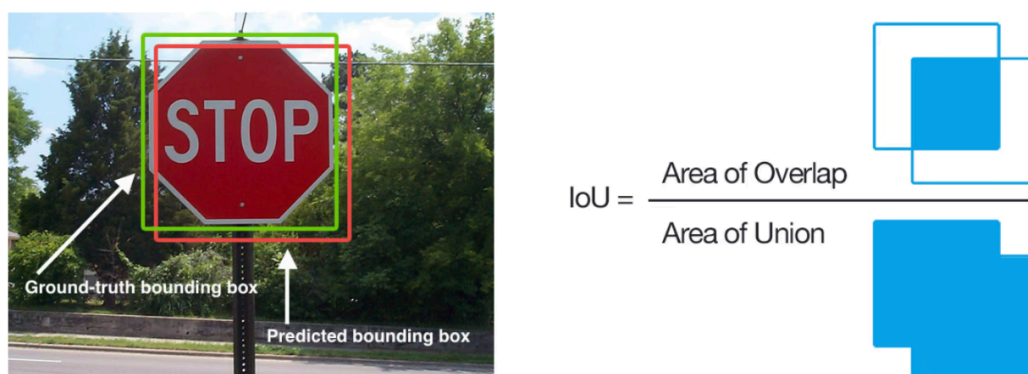


Figura 36. Càlcul de l'Intersection over Union (IOU)

Consisteix a dividir l'àrea que les dues caixes comparteixen, entre l'àrea conjunta d'ambdues caixes (*figura 36*).

El resultat serà un valor comprès entre 0 i 1. Si és proper a 0, significa que gairebé no hi ha superposició entre les caixes, mentre que si és proper a 1, indica que les caixes se superposen gairebé completament.

- **Confusion Matrix**

És una matriu on s'avaluen les classes de prediccions que un model ha realitzat (figura 37).

		Predicted Class	
		Positive	Negative
Actual Class	Positive	TP	FN
	Negative	FP	TN

Figura 37. Explicació d'una matriu de confusió

En un sistema de reconeixement de senyals de trànsit, la matriu de confusió permet identificar amb precisió quins tipus de senyals i situacions generen errors de classificació. Aquesta anàlisi ajuda els desenvolupadors a detectar les debilitats del model i a enfocar el reentrenament en les classes de senyals o condicions que presenten més dificultats, millorant d'aquesta manera el seu rendiment global.

- **ROC i AUC**

La corba ROC (*Receiver Operating Characteristic*) és una gràfica que mostra el rendiment d'un model de classificació binària en diferents llindars de decisió (*threshold*). Ens expressa com de bé un model pot discriminar entre classes positives i negatives, i com el podem ajustar per optimitzar el rendiment.

Tenim a l'eix **X** el **FPR** (1-Especificitat) i en l'eix **Y** el **recall** (*Sensitivity* o TPR) d'un model:

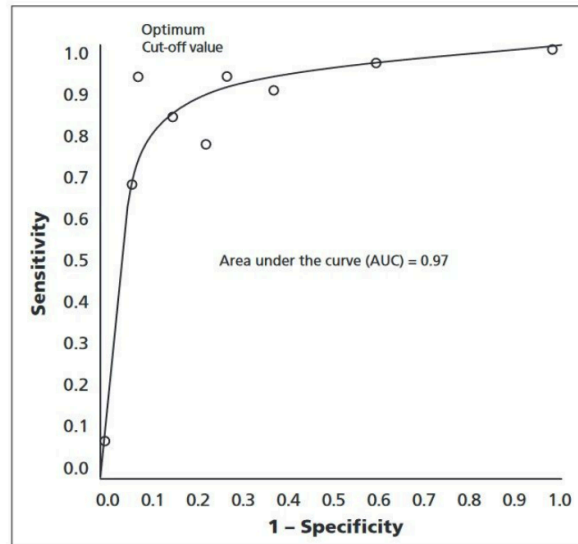


Figura 38. Exemple de corba ROC i AUC

A través de diferents llindars, obtindrem la proporció de TPR i FPR el que ens donarà els valors de la corba ROC.

El llindar òptim en una corba ROC sovint es coneix com a punt d'Youden. Aquest és el punt de la corba ROC que ofereix el millor compromís entre detectar correctament els positius (alta TPR) i minimitzar els falsos positius (baixa FPR). Aquest punt sol estar a prop de la cantonada superior esquerra de la corba ROC.

L' **AUC** (*Area Under the Curve*) representa l'àrea per sota de la corba ROC. És la integral de la corba ROC, i el valor correspon entre 0 i 1. Aquest valor resumeix el rendiment total del model per a tots els llindars. Un valor d'AUC alt indica un major rendiment del model, és a dir, una correcta discriminació entre classes positives en una varietat de llindars diferents.

- **PR Curve (Precision-Recall Curve)**

Com la corba ROC, també tenim una corba, la qual mostra la relació entre la **precisió** i el **recall** del model a diferents llindars de decisió. Ens és útil sobretot per a casos on les classes no estan gaire equilibrades.

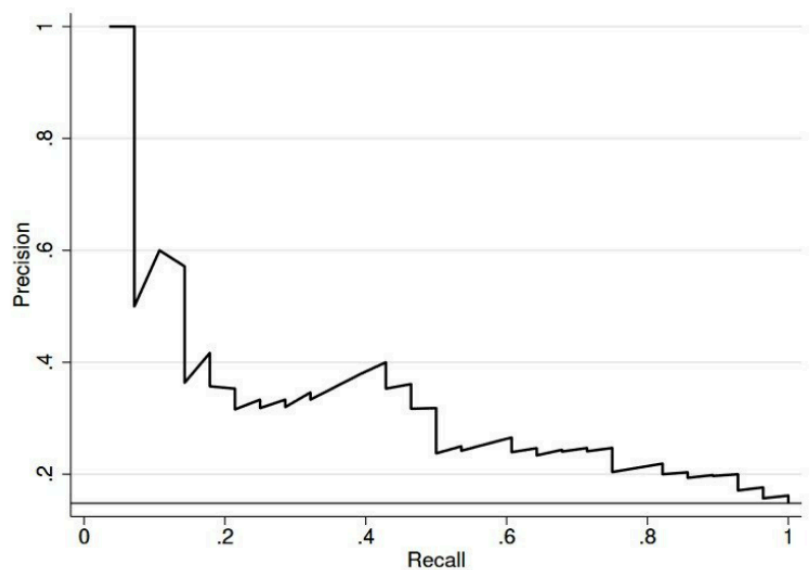


Figura 39. Exemple d'una corba Precision-Recall (PR)

Una corba PR ideal es situaria prop de la cantonada superior dreta (alta precisió i alt *recall*), indicant que el model és capaç de detectar gairebé tots els casos positius (alt *recall*) amb molt pocs falsos positius (alta precisió).

- **Average Precision (AP)**

Consisteix a calcular l'àrea sota la corba Precision-Recall. És una mètrica que resumeix el rendiment global del model. És especialment útil en situacions on les classes són desbalancejades, és a dir, quan algunes classes tenen molts menys exemples que d'altres.

A diferència de la precisió i el *recall*, l'AP captura millor el rendiment del model en aquestes condicions, ja que té en compte tota la corba PR en lloc de fixar-se en un sol punt de llindar. Un valor proper a 1 indica que el model té tant una alta precisió com un alt *recall* per als diferents llindars.

$$AP = \int_0^1 P(r) dr$$

, on $P(r)$ és la precisió en funció del *recall*.

- **Mean Average Precision (mAP)**

El mAP (*mean Average Precision*) és una mètrica molt utilitzada per avaluar el rendiment d'un model en tasques com la detecció d'objectes, especialment quan s'analitza una corba PR.

El mAP calcula l'AP per a cada classe i després en fa la mitjana, donant una mètrica global que reflecteix el rendiment del model en totes les classes. És una mètrica comuna per calcular la precisió d'un *dataset* multiclasse per un model.

$$mAP = (1/N) \sum_{k=1}^{k=n} AP_k$$

, on N és el nombre total de classes i AP_k l' *average precision* de la classe k .

El mAP permet identificar les classes en què el model té un rendiment inferior. Aquesta informació ens és útil per detectar quines classes estan desequilibrades o presenten més dificultats per al model.

8. Experiments i Resultats

En aquest apartat, entrenarem diversos models de YOLO per detectar i classificar senyals de trànsit. Posteriorment, avaluarem el rendiment dels models utilitzant les mètriques explicades a l'apartat 7.

Hem realitzat un total de tres experiments. En cada un d'ells, hem entrenat o preentrenat (*fine-tuning*) un model de YOLO amb els diferents conjunts de dades, exposats al punt 6.

Per a l'avaluació de cada un dels models realitzarem una **fase de test** amb un conjunt d'imatges que el model no haurà vist mai. Aquest conjunt serà el de *validation*.

8.1. Model A

Pel primer experiment, partirem d'un model preentrenat.

Normalment, quan s'aborda un problema de visió artificial com el nostre, no es comença amb un model completament verge, ja que això requeriria molt de temps en ajustar els pesos de la CNN per aconseguir un rendiment inicial adequat.

Utilitzarem els pesos d'un model entrenat amb COCO (un *dataset* no específic sobre el problema de detecció de senyals que volem tractar) per aprofitar el coneixement general adquirit en altres tasques, reduint molt el temps d'entrenament.

Un cop tenim el model A preparat, volíem avaluar-lo amb el conjunt EuRoSign. Tal com s'explica al problema del punt 9.1.2, no podem avaluar un model entrenat amb un conjunt de 80 classes, amb un dels nostres *datasets* de 4 classes. Per tant, he considerat que es podria realitzar un *fine-tuning* d'una sola època, amb el conjunt EuRoSign, per tal que podem després avaluar el rendiment del model amb un dels *datasets* de senyals de trànsit.

Model A = Model COCO + Fine-tuning d'una època d'EuRoSign

8.2. Model B

En aquest segon experiment hem volgut adaptar el model per al nostre objectiu.

Hem realitzat un *fine-tuning* de 20 èpoques, del model A amb el *dataset* EuRoSign.

Model B = Model A + Fine-tuning de 20 èpoques d'EuRoSign

Les imatges del *dataset* d'EuRoSign pertanyen a Europa, per tant, volem veure quin rendiment ofereix aquest model en detecció i classificació de senyals europees.

8.3. Model C

L'objectiu d'aquest darrer experiment és adaptar el model B per fer prediccions sobre imatges d'Alemanya. Per tant, farem un *fine-tuning* de 100 èpoques sobre el model B del conjunt GTSDb.

Model C = Model B + Fine-tuning de 100 èpoques de GTSDb

Un cop fets els tres experiments, voldrem veure si tenim un problema de ***catastrophic forgetting***. Aquest problema esdevé quan un model perd coneixement anterior en intentar aprendre nova informació.

En el nostre cas voldrem veure si un model entrenat primer amb el conjunt d'Europa i després amb el d'Alemanya (model C), perd rendiment si l'avaluem de nou amb el conjunt d'Europa.

8.4. Resultats Qualitatius

Els resultats qualitatius serveixen per analitzar visualment el rendiment del model. Mostraré exemples d'imatges amb les deteccions predites per observar si el model identifica correctament els senyals. Aquesta part complementa l'anàlisi quantitatiu.

En realitzar l'avaluació dels models he obtingut un fitxer JSON amb les deteccions predites per cada imatge. Mitjançant un script de *python* (annex X), he dibuixat les deteccions a les imatges i les he comparat amb les respectives etiquetes.

Cada detecció és causada per un nivell de confiança. He considerat totes aquelles deteccions per sobre d'un llindar de confiança del 0.3. No he considerat un llindar

més elevat perquè hi ha deteccions amb confiança baixa a causa de les condicions de llum extremes, diferents angles del senyal, etc.

Model A avaluat amb el conjunt EuRoSign

Casos d'èxit:



Figura 40 (a). Predicció correcta per la imatge 00473.jpg



Figura 40 (b). Imatge 00473.jpg etiquetada



Figura 41 (a). Predicció correcta per la imatge 00417.jpg



Figura 41 (b). Imatge 00417.jpg etiquetada

En les dues figures 40 i 41 podem veure una correcta detecció i classificació d'alguns senyals. Com ja hem comentat anteriorment, les condicions de llum en aquest conjunt són força extremes. En totes dues imatges, però sobretot a la 41, hi ha un contrast molt marcat entre el cel i els senyals, fet que redueix la seguretat del model en la predicció, i provoca un nivell de confiança baix.



Figura 42 (a)c. Predicció correcta per la imatge 00393.jpg



Figura 42 (b)c. Imatge 00393.jpg etiquetada

En alguns casos, com en la figura 42, la distància de la càmera al senyal és gran, el que fa que la resolució del senyal sigui molt petita i el model tingui dificultats per predir-la.

Aquest model ha estat entrenat només amb una època d'un conjunt de dades específic de senyals. Per la qual cosa pot sorprendre el seu bon rendiment malgrat el poc temps d'entrenament. Ara bé, també hi trobem casos problemàtics.

Casos problemàtics:



Figura 43 (a). Predicció incorrecta per la imatge 00445.jpg



Figura 43 (b). Imatge 00445.jpg etiquetada

A la figura 43 es poden veure dues errades, les quals reflecteixen la baixa fiabilitat del model. Observem com el model ha confós un triangle blanc amb contorn vermell amb un senyal *other*. La confiança d'aquesta detecció és molt baixa.

Observem que no s'ha detectat el senyal de la dreta del tot. Pot haver sigut a causa de l'angle del senyal, ja que no està ben perpendicular a la càmera. Aquest és un factor que també pot dificultar la detecció, a més de les baixes condicions de lluminositat.



Figura 44 (a). Predicció incorrecta per la imatge 00369.jpg

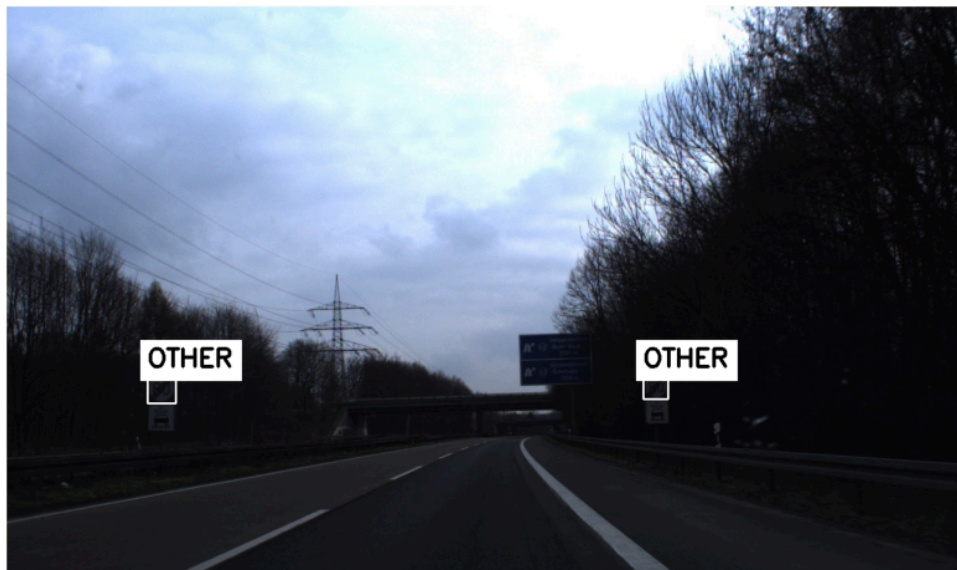


Figura 44 (b). Imatge 00369.jpg etiquetada

A la figura 44, tenim un error específic del model de YOLO. Anteriorment, havia explicat que aquest model calcula N caixes delimitadores per cel·la. Aquestes N caixes normalment pertanyen al mateix objecte. Finalment, es trobava la millor caixa delimitadora amb la tècnica del *non max suppression*. En aquest cas, tenim un problema d'*Overlapping detections*, que es produeix quan no s'aconsegueix eliminar adequadament les deteccions redundants, deixant més d'una caixa delimitadora per a l'objecte (figura 44 (a)).



Figura 45 (a). Predicció incorrecta per la imatge 00351.jpg



Figura 45 (b). Imatge 00351.jpg etiquetada



Figura 46 (a). Predicció incorrecta per la imatge 00396.jpg



Figura 46 (b). Imatge 00396.jpg etiquetada



Figura 47 (a). Predicció incorrecta per la imatge 00526.jpg



Figura 47 (b). Imatge 00526.jpg etiquetada



Figura 48 (a). Predicció incorrecta per la imatge 00405.jpg

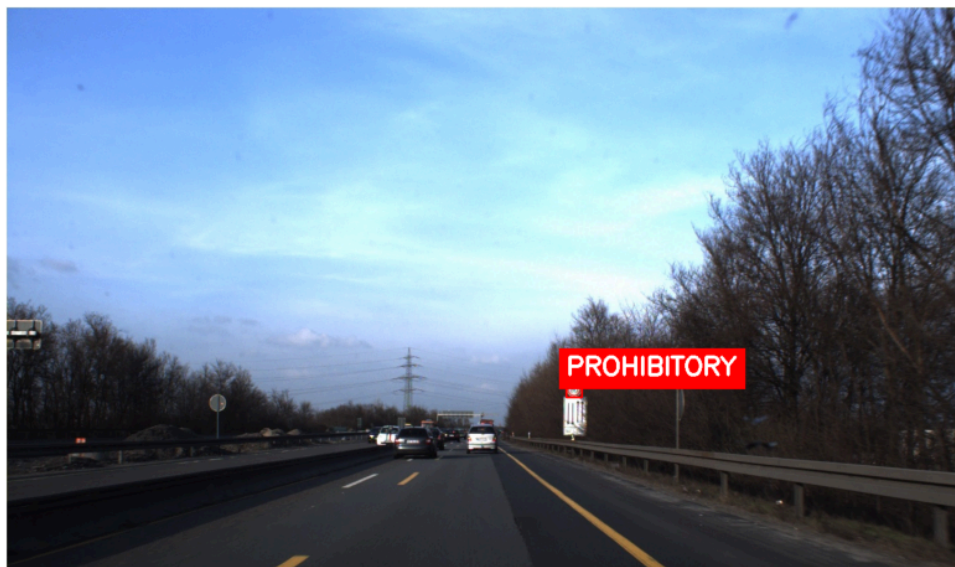


Figura 48 (b). Imatge 00405.jpg etiquetada

Si ens fixem en aquestes darreres figures (*figures 45, 46, 47 i 48*) trobem que les deteccions més errònies pertanyen a la classe *other*. La majoria d'aquests errors són **falsos positius**, és a dir, deteccions d'objectes semblants a senyals que no ho són. Bé, en el cas de la figura 48, la predicció de la dreta sí que ho és, però el senyal és pel sentit contrari, per tant, no es considera una predicció correcta.

A l'apartat dels resultats quantitatius veure'm numèricament que molts dels casos de falsos positius pertanyen a la classe *other* i la possible causa.

Model B avaluat amb el conjunt EuRoSign

Casos d'èxit:



Figura 49 (a). Predicció correcta per la imatge 00526.jpg



Figura 49 (b). Imatge 00526.jpg etiquetada



Figura 50 (a). Predicció correcta per la imatge 00405.jpg



Figura 50 (b). Imatge 00405.jpg etiquetada

Tant la imatge de la figura 49 com la 50, es prediu el dors de les senyals pel model A. Podem veure com pel model B això ja no ocorre.

Casos problemàtics:



Figura 51 (a). Predicció incorrecta per la imatge 00002.jpg



Figura 51 (b). Imatge 00002.jpg etiquetada

A la primera imatge del conjunt de validació (*figura 51*) podem observar un cas de fals negatiu poc habitual. Costa d'entendre que el model no hagi predit correctament el senyal. Les condicions de llum són bones i la qualitat de la imatge també. El senyal està pràcticament perpendicular, i hi ha poca distància des de la càmera.

Per trobar una explicació, he preparat un *script* el qual mostra totes les imatges del conjunt de validació amb senyals d'obligació (*figura 52*).

00453.jpg



00039.jpg



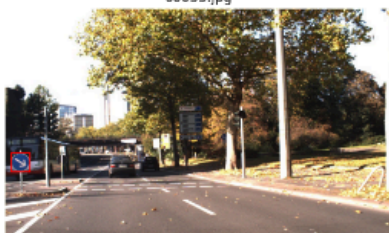
00430.jpg



00066.jpg



00033.jpg



00084.jpg



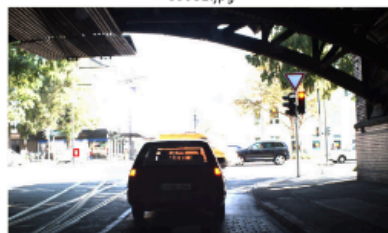
00250.jpg



00210.jpg



00081.jpg



00054.jpg



00060.jpg



00451.jpg





Figura 52. Imatges amb senyals d'obligació del conjunt de validació EuRoSign

He realitzat un recompte manual dels senyals d'obligació en funció del sentit on apunten. He trobat 17 senyals que apunten a la dreta, 5 senyals que apunten cap

endavant i només 3 que apunten cap a l'esquerra, de les quals 1 també apunta cap endavant.

Trobem que 1 de cada 10 senyals d'obligació són senyals cap a l'esquerra, el que fa que el model predigui erròniament aquests casos. Un *dataset* amb més mostres de senyals d'una mateixa classe, milloraria el rendiment del model.



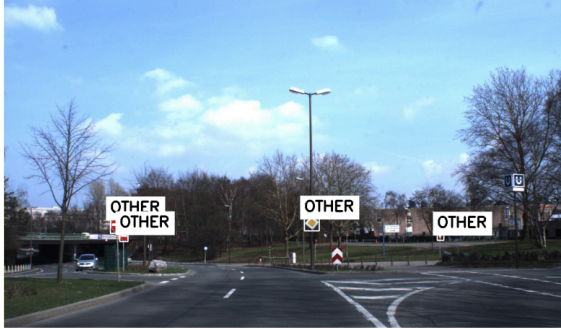
Figura 53 (a). Predicció incorrecta per la imatge 00451.jpg



Figura 53 (b). Imatge 00451.jpg etiquetada

En el cas de la figura 53, trobem que el model distingeix un senyal de pàrquing per una de *mandatory*. I és que observant les deteccions *fals positives* d'aquesta

avaluació, trobem que gran part d'aquests senyals no són considerades (al punt 6 es contemplen els senyals que no són considerats).

Model A	
 Figura 54 (a). Predicció incorrecta per la imatge 00445.jpg	 Figura 54 (b). Imatge 00445.jpg etiquetada
Model B	
 Figura 55 (a). Predicció incorrecta per la imatge 00445.jpg	 Figura 55 (b). Imatge 00445.jpg etiquetada

Taula 5. Comparativa de prediccions sobre la imatge 00445.jpg pel conjunt A i B.

En general, per l'avaluació del model B trobem una millora en el rendiment respecte al model A. Ara bé, hi ha casos on l'entrenament de 20 èpoques no és suficient per obtenir una predicció excel·lent. És el cas de la figura de la taula 5, que si la comparem amb l'avaluació del model A, veiem que ja no prediu el senyal *danger* que no hi era, però encara no prediu el senyal *other* de la dreta.

Model C avaluat amb el conjunt GTSDb

Casos d'èxit:

Model A



Figura 56 (a). Predicció incorrecta per la imatge 00369.jpg

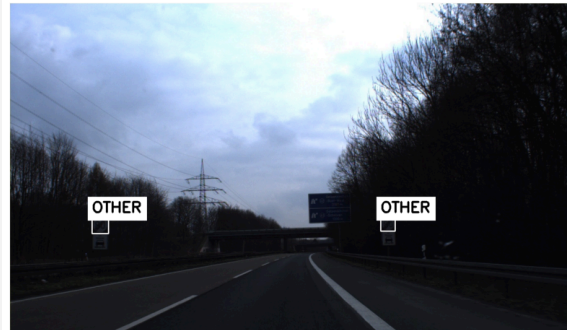


Figura 56 (b). Imatge 00369.jpg etiquetada

Model B



Figura 57 (a). Predicció incorrecta per la imatge 00369.jpg

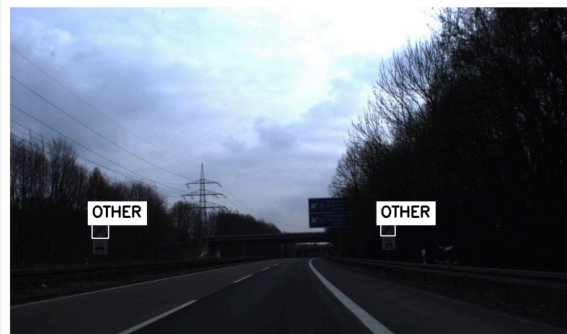


Figura 57 (b). Imatge 00369.jpg etiquetada

Model C

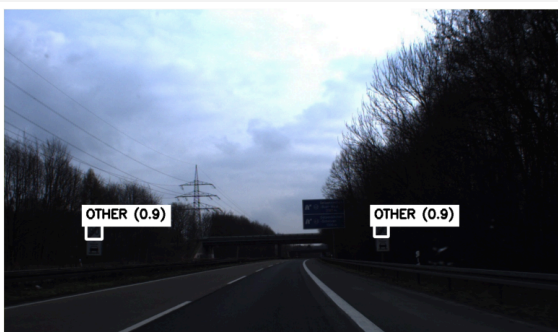


Figura 58 (a). Predicció correcta per la imatge 00369.jpg

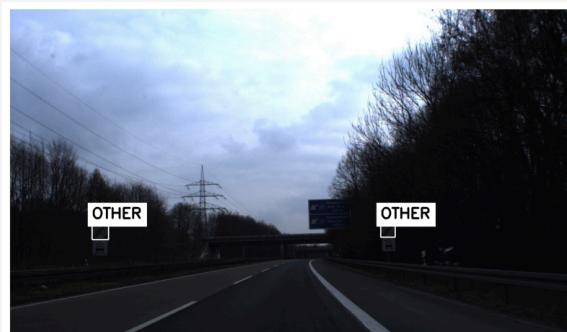
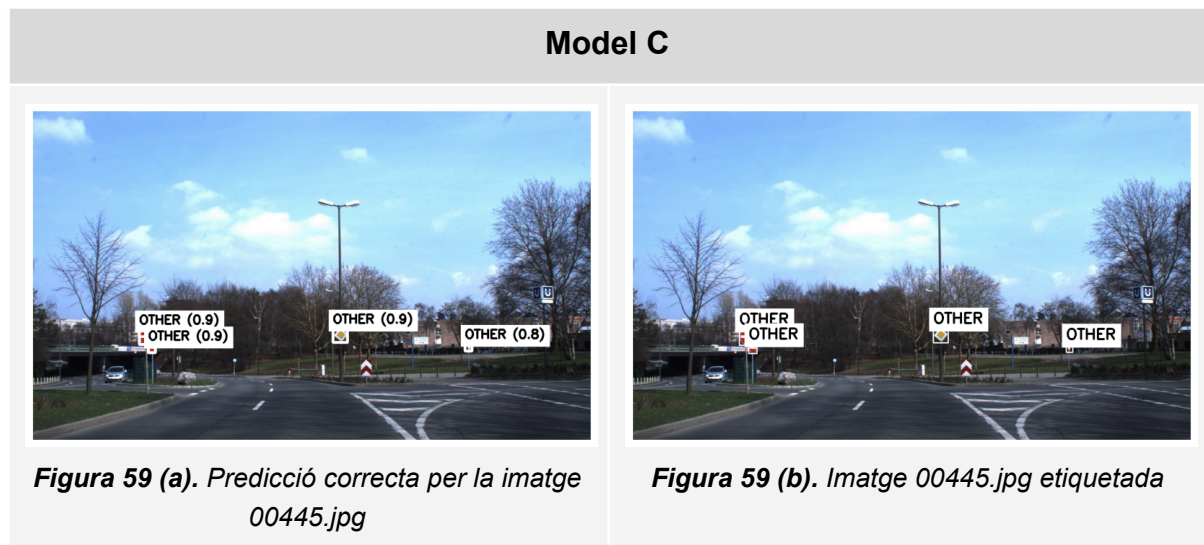


Figura 58 (b). Imatge 00369.jpg etiquetada

Taula 6. Comparativa de prediccions sobre la imatge 00369.jpg dels models A, B i C.

A la taula 6 podem observar la correcta predicció del model C amb una imatge amb poca lluminositat.



Taula 7. Comparativa de prediccions sobre la imatge 00445.jpg pel model C.

Seguint l'evolució de la imatge 00445.jpg amb els tres models (*taula 5 i 7*), finalment obtenim una predicció correcta de la imatge amb el model C.



Figura 60. Predicció correcta per la imatge 00002.jpg

A la figura 60 trobem que el model C sí que és capaç de predir el senyal de la imatge 00002.jpg amb plena confiança.

Cal destacar que la figura 60 i les figures de les taules 6 i 7, són del *dataset* EuRoRoad i el model C l'hem obtingut fent un *fine-tuning* del GTSDDB. Per tant, podem veure visualment com de bé ha predit aquests casos tan complicats, d'un *dataset* diferent a l'últim que s'ha entrenat.

Ara mostraré alguns casos d'èxit pel *dataset* GTSDDB.



Figura 61 (a). Predicció correcta per la imatge 00447.jpg

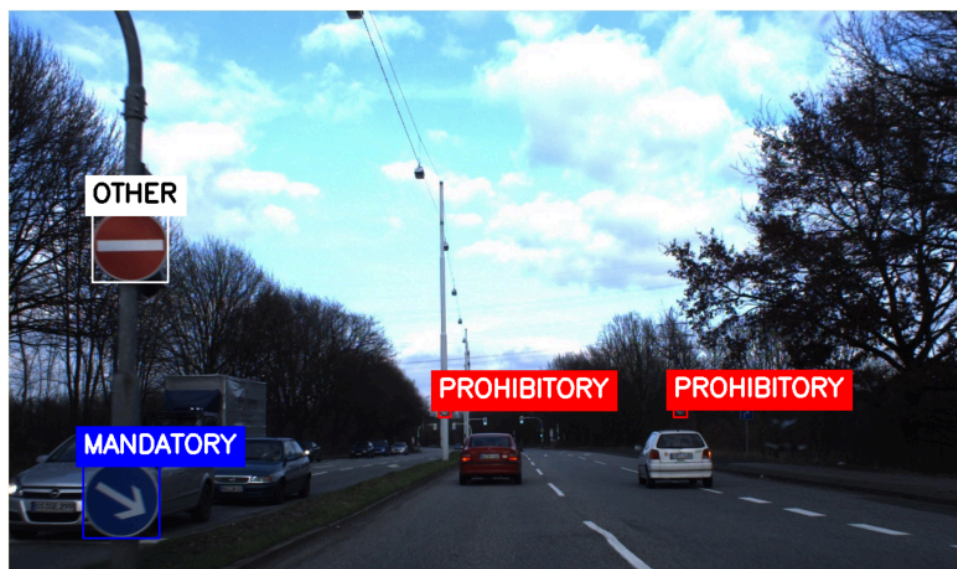


Figura 61 (b). Imatge 00447.jpg etiquetada



Figura 62 (a). Predicció correcta per la imatge 00531.jpg



Figura 62 (b). Imatge 00531.jpg etiquetada

En les últimes figures 61 i 62 podem veure com de bé el model és capaç de predir els senyals més propers a la càmera, i també les més allunyades amb una confiança elevada.

Casos problemàtics



Figura 63 (a). Predicció incorrecta per la imatge 00549.jpg



Figura 63 (b). Imatge 00549.jpg etiquetada



Figura 64 (a). Predicció incorrecta per la imatge 00485.jpg



Figura 64 (b). Imatge 00485.jpg etiquetada

Un problema que encara ocorre amb el model C és la detecció de senyals del sentit contrari (*figures 63 i 64*). Una causa pot ser que s'ha entrenat senyals on la il·luminació és tan baixa, amb tan poc contrast de color, que el model ho confon amb el dors gris d'un senyal. Però en aquesta última figura només es detecten dos dels 4 senyals del sentit contrari.



Figura 65 (a). Predicció incorrecta per la imatge 00575.jpg



Figura 65 (b). Imatge 00465.jpg etiquetada



Figura 66 (a). Predicció incorrecta per la imatge 00465.jpg



Figura 66 (b). Imatge 00465.jpg etiquetada

Un problema que m'he topat és que hi ha imatges, com el cas de les figures 65 i 66, on el model detecta un senyal, el qual sí que hi és, però no està etiquetada.

Aquest conjunt de dades, GTSDDB, és considerat molt fiable, i és molt usat per empreses del sector. Una possibilitat és que aquests senyals tinguin una resolució tan baixa, o estan tan allunyades de la càmera que el conjunt no les té en compte. De fet, a ull nu costa identificar la classe a la qual pertanyen.

En definitiva, es veu una evolució positiva en el rendiment dels diferents models. Aquesta evolució és destacada entre el model B i el C, amb un entrenament de 20 i 100 èpoques, respectivament.

Finalment, m'agradaria comparar els senyals dels dos conjunts per veure com són de similars (*taula 8*).



Taula 8. Comparativa de senyals entre el conjunt EuRoSign i el GTSDb

Comparant ambdós conjunts, observem que els senyals són molt similars entre sí. La majoria dels senyals d'Alemanya tenen un contorn blanc, mentre que els senyals europeus poden tenir-lo o no. Pel que fa a la resta de factors, com la forma o el disseny dels senyals, són pràcticament idèntiques.

8.5. Resultats Quantitatius

És important definir una estratègia de validació adequada pel nostre problema per saber quines mètriques de les mencionades al punt 7 són més importants i ens donen més informació.

En aquest projecte no s'està realitzant la detecció i classificació de senyals de trànsit per a cap aplicació en específic. De totes maneres, podem adoptar el context que s'exposa la visió artificial aplicada a la conducció autònoma.

En aquest context, el cost que el model se li escapi un senyal de prohibició és molt elevat (hi poden haver costos humans). Per tant, ens interessa un valor elevat de *recall* (proporció de deteccions correctes amb el total d'objectes). Ara bé, també ens interessa una alta precisió (proporció de deteccions correctes amb el total de deteccions fetes), ja que no volem fer deteccions incorrectes com detectar un objecte inofensiu com un obstacle. És per això que volem un **model equilibrat en precisió i recall**.

Les mètriques que ofereixen aquesta informació, a part de la precisió i *recall*, són la corba *Precision-Recall* amb el *mean Average Precision* i el *F1 Score*.

Model A avaluat amb el conjunt EuRoSign

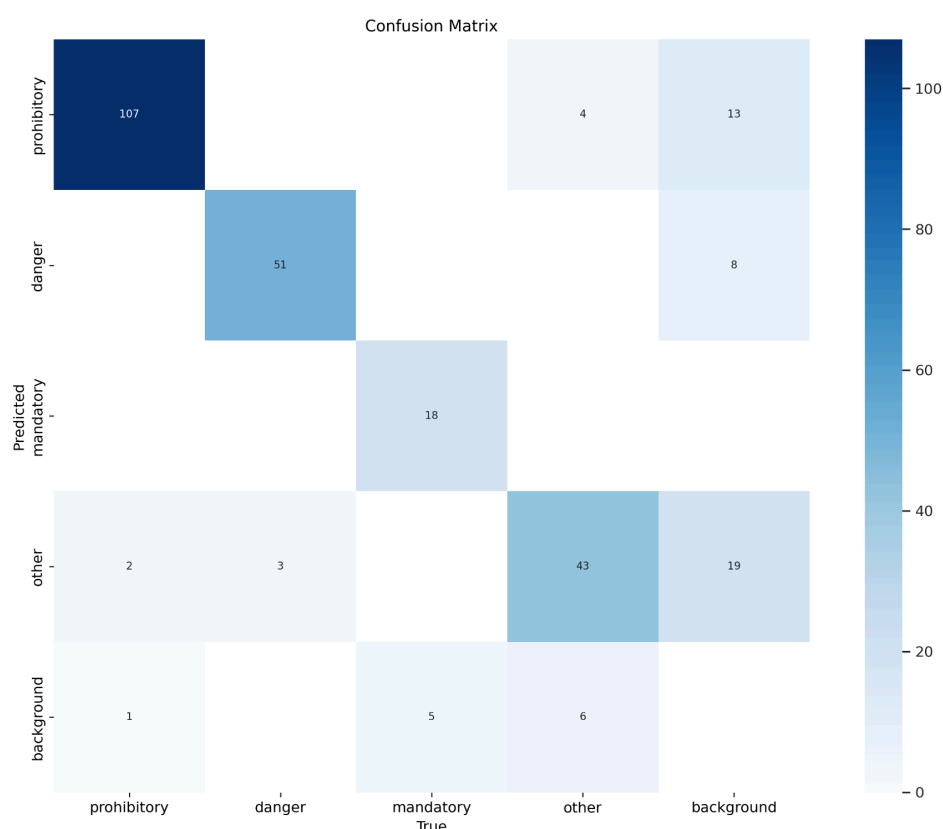


Figura 68. Matriu de confusió no normalitzada del model A per l'avaluació amb EuRoSign

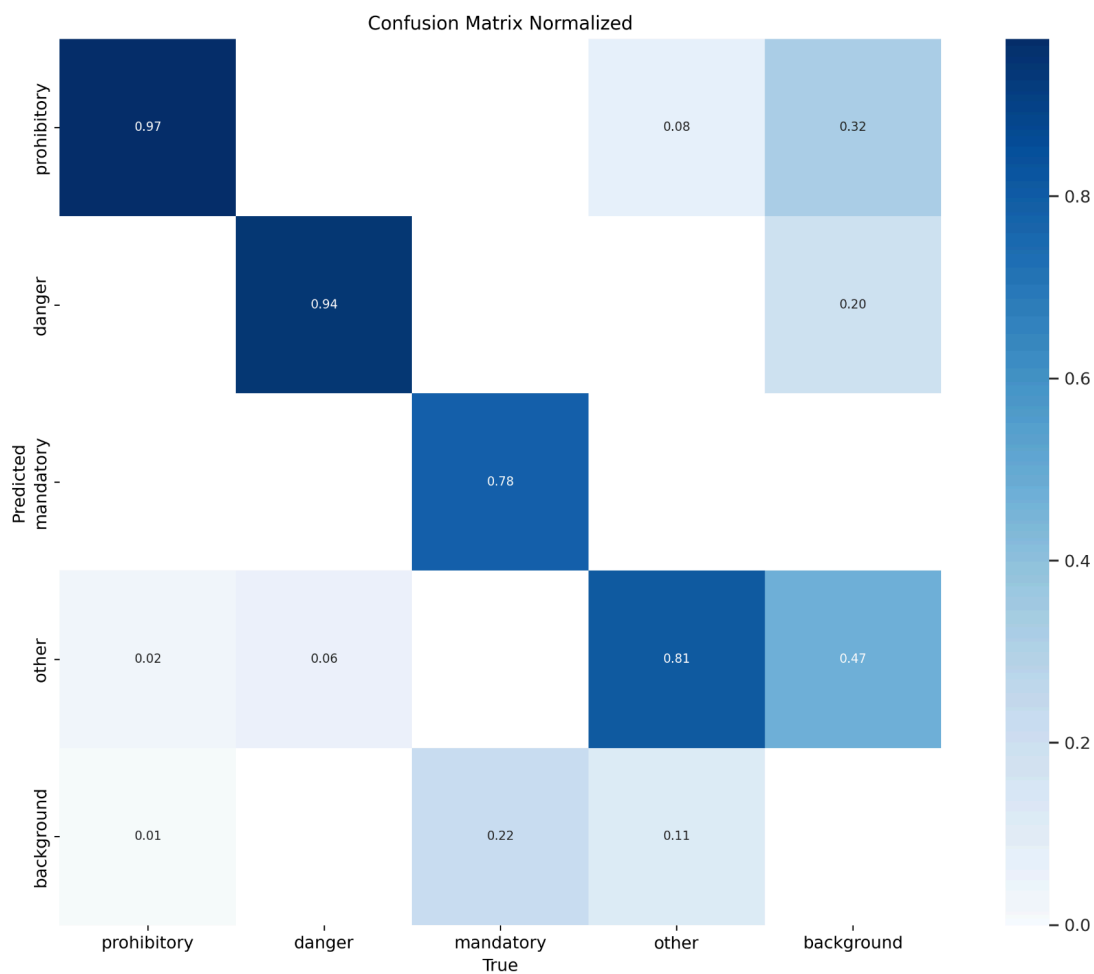


Figura 69. Matriu de confusió normalitzada del model A per l'avaluació amb EuRoSign

Amb la matriu de confusió (figures 68 i 69) podem visualitzar la capacitat del model a l'hora de classificar les deteccions.

	Deteccions Totals	Deteccions classificades correctament	Deteccions classificades incorrectament	<i>False Positives</i>	<i>True Negatives</i>
Model A	280	219 (78,2%)	9 (3,2%)	40 (14,3%)	12 (4,3%)

Taula 9. Desglossament de les dades de la matriu de confusió pel model A amb l'avaluació d'EuRoSign

Al desglossar les dades de les matrius de confusió a la taula 9, trobem que el 78,2% de les deteccions s'han realitzat correctament. Ara bé, també destaca l'alt percentatge de falsos positius.

Gairebé la meitat d'aquests falsos positius es prediuen com a senyals *other*, mentre que el 32%, es prediuen com a senyals de prohibició.

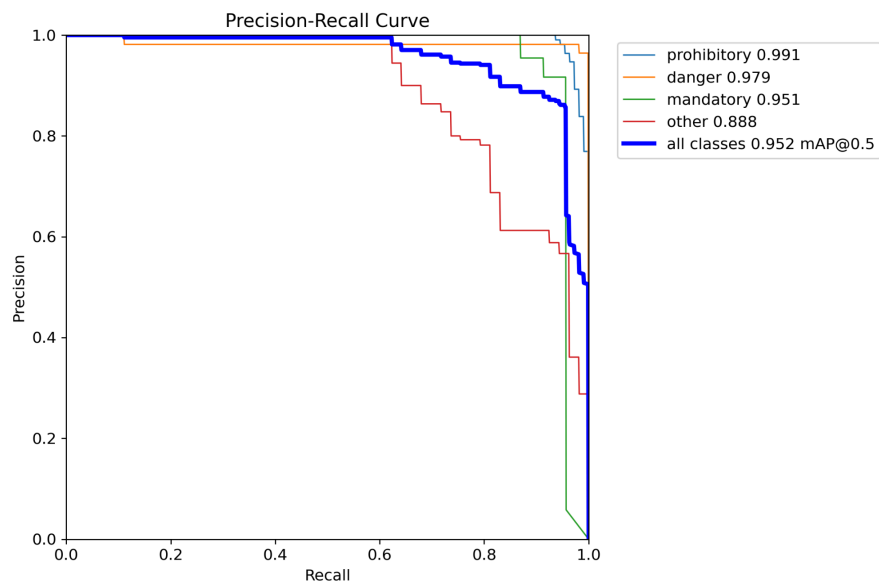


Figura 70. Gràfica precision-recall del model A per l'avaluació amb EuRoSign

A la figura 70, podem observar com les deteccions de la classe *other* estan molt descompensades amb relació a les altres classes. Un 36% de les deteccions de la classe *other* s'han classificat en altres classes, sobretot en *background*, és a dir, que no hi ha senyals.

En general, té sentit haver detectat una gran quantitat d'objectes que no són senyals (falsos positius), perquè el model A parteix d'un entrenament de COCO, on es fan deteccions d'elements quotidians que poden aparèixer a les imatges dels conjunts de senyals.

A l'apartat dels resultats qualitatius, hem vist que aquestes deteccions són d'objectes rodons amb una forma similar a la dels senyals, per tant, puc entendre l'error del model, el qual la majors part del seu coneixement no és sobre senyals.

Model B avaluat amb el conjunt EuRoSign

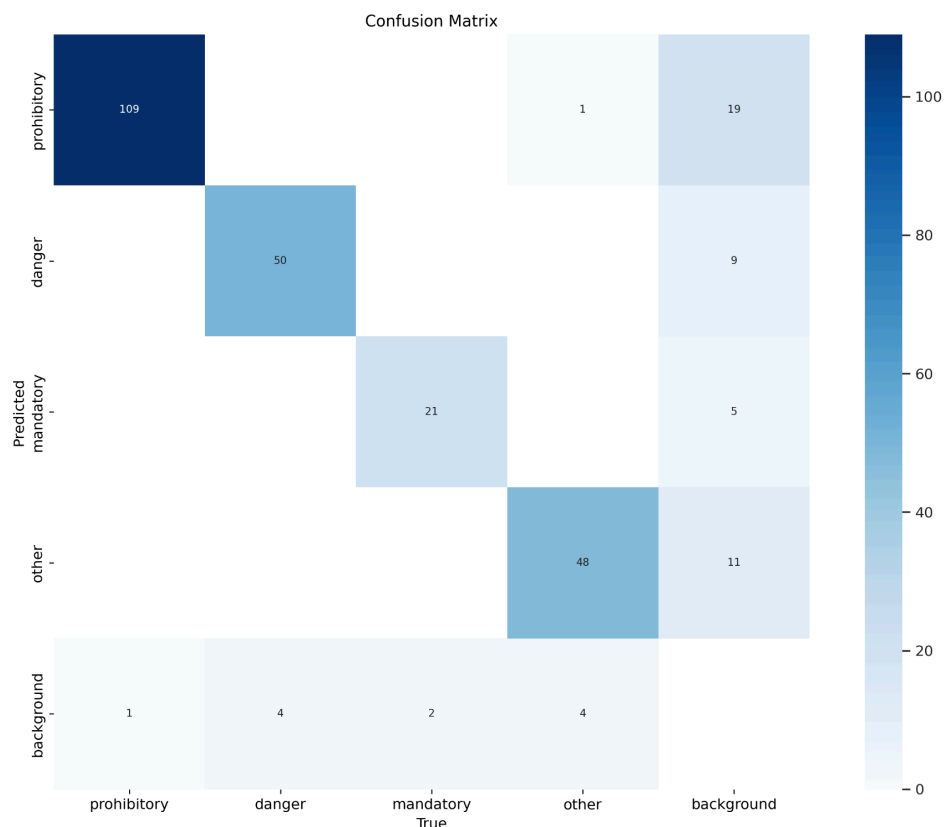


Figura 71. Matriu de confusió no normalitzada del model B per l'avaluació amb EuRoSign

Si comparem les matrius de confusió dels models A i B veiem com hi ha una clara millora en la classificació de les deteccions (diagonal principal de la matriu).

	Deteccions Totals	Deteccions classificades correctament	Deteccions classificades incorrectament	<i>False Positives</i>	<i>True Negatives</i>
Model A	280	219 (78,2%)	9 (3,2%)	40 (14,3%)	12 (4,3%)
Model B	284	228 (80,3%)	1 (0,3%)	44 (15,5%)	11 (4,4%)

Taula 10. Desglossament de les dades de la matriu de confusió pel model A i B per l'avaluació amb EuRoSign

A la taula 10, podem observar com els valors d'ambdós models són bastant similars, excepte el nombre de deteccions classificades incorrectament. El model B s'ha entrenat amb 20 èpoques, cosa que ha fet augmentar la capacitat de classificació.

Veiem com augmenten les deteccions correctes un 2%, però es manté o inclús empitjora la proporció de falsos positius i veritables negatius.

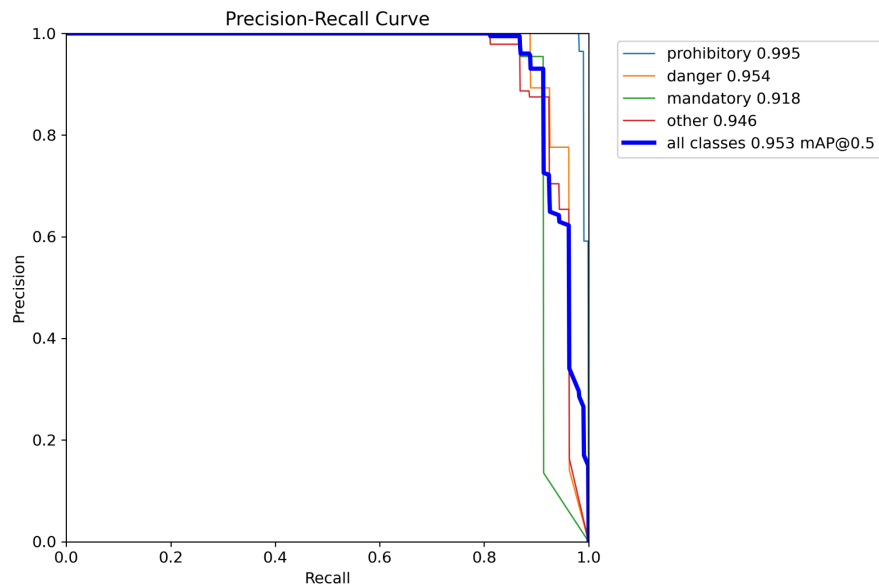


Figura 72. Gràfica precision-recall del model B per l'avaluació amb EuRoSign

A la figura 72 es veu clarament com la precisió de les deteccions de senyals *other* ha augmentat, la qual cosa s'igualava amb la resta de classes, però trobem que els senyals d'obligació amb un alt recall perden precisió.

En les validacions dels dos models hem vist que, pels models A i B, només un 8% i 9% respectivament, són deteccions de senyals d'obligació. A l'apartat 6.2 podem observar com la classe *mandatory* és la més escassa en aquest subconjunt.

Per tant, pensem que la falta de precisió en la classe *mandatory* es podria corregir amb un conjunt més balancejat de classes. És el cas dels senyals de prohibició on la quantitat de mostres dobla a les de les altres classes. Observem a la figura Xi el rendiment dels senyals de prohibició el qual és quasi perfecte.

Model C avaluat amb el conjunt GTSDb

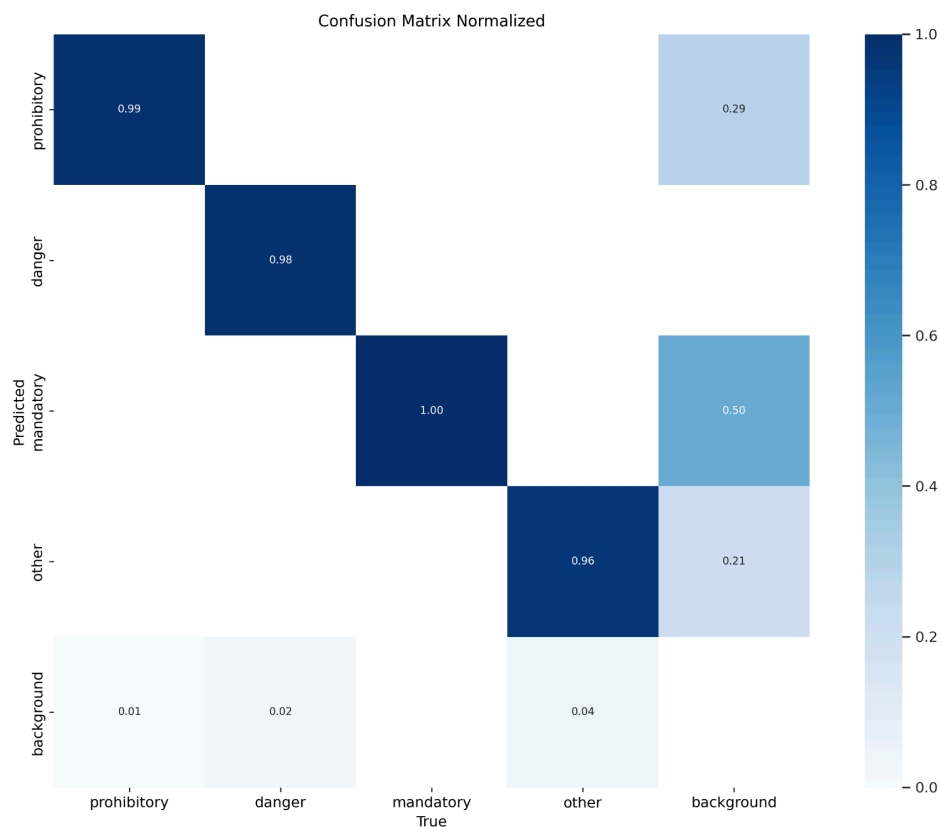


Figura 73. Matriu de confusió normalitzada del model C per l'avaluació amb GTSDb

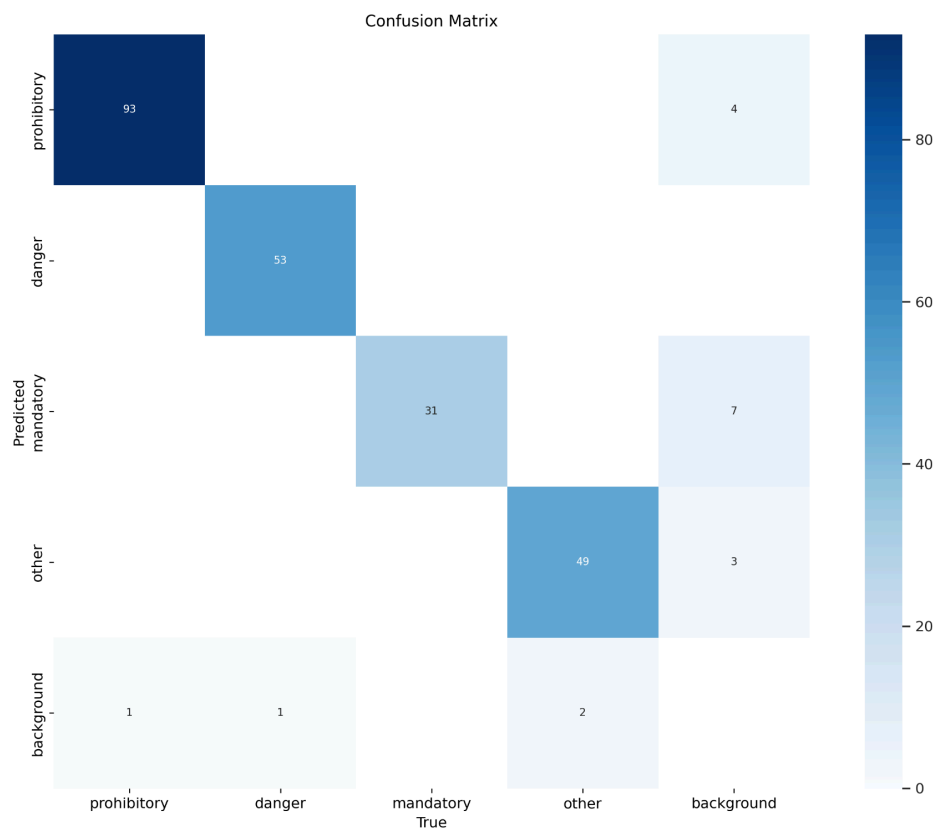


Figura 74. Matriu de confusió no normalitzada del model C per l'avaluació amb GTSDb

	Deteccions Predites	Deteccions Reals	Deteccions classificades correctament	Deteccions classificades incorrectament	False Positives	True Negatives
Model A	280	241 _{EuRoSign}	219 (78,2%)	9 (3,2%)	40 (14,3%)	12 (4,3%)
Model B	284	241 _{EuRoSign}	228 (80,3%)	1 (0,3%)	44 (15,5%)	11 (4,4%)
Model C	244	230 _{GTSDb}	226 (92,6%)	0 (0%)	14 (5,7%)	4 (1,6%)

Taula 11. Comparativa dels models A, B i C, per l'avaluació amb GTSDb

Mitjançant les matrius de confusió (*figures 74 i 75*) i afegint les dades del model C a la taula 11, podem observar la clara millora en el model després d'un entrenament de 100 èpoques.

De les 230 deteccions reals, 226 són detectades i classificades correctament, per tant, les 4 restants han sigut prediccions no predites (*True Negatives*).

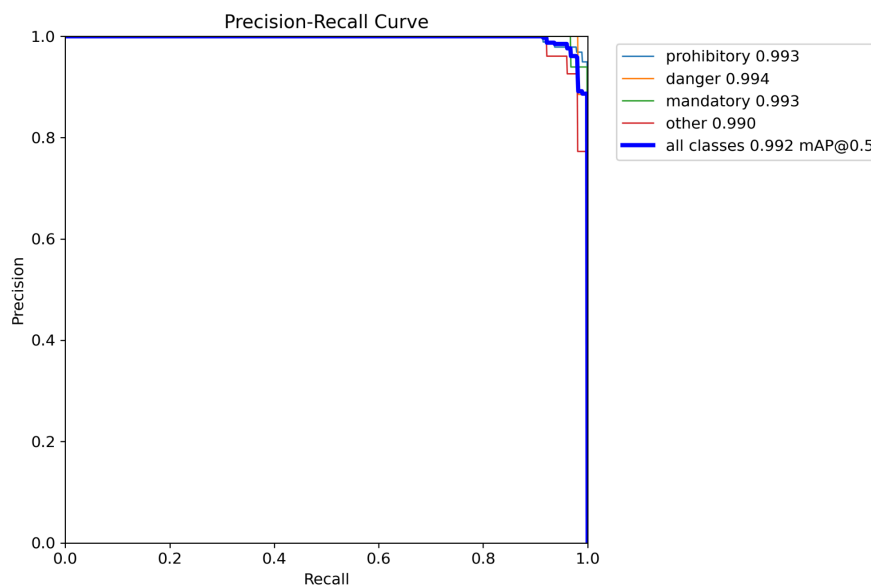


Figura 75. Gràfica precision-recall del model C

En aquest últim experiment, s'ha realitzat un entrenament de 100 èpoques. Si observem la figura 75 veiem que tant la precisió com el *recall* són elevats, el que fan un model ben balancejat.

A diferència de la figura 72 per un alt *recall* tenim una bona precisió. El nombre de falsos positius ha disminuït significativament. Si la precisió és la proporció de deteccions correctes del total de deteccions fetes, trobem que a l'haver detectat menys falsos positius hem millorat el model. En aquest model es faran les deteccions necessàries (*recall* elevat) per tenir una precisió elevada.

8.6. Comparativa de resultats qualitatius i quantitatius

Una pregunta que ens vàrem fer a l'inici del treball va ser si un model perd rendiment a mesura que se li apliquen *fine-tunings*. És a dir, si hem tingut el problema de *catastrophic forgetting* o no pel model C.

	Test EuRoSign		Test GTSDb	
	mAP@0.5	F1 _(Llindar)	mAP@0.5	F1 _(Llindar)
Model A (COCO)	0.952	0.89 _(0.404)	0.953	0.88 _(0.250)
Model B (ft EuRoSign)	0.953	0.93 _(0.683)	0.932	0.90 _(0.759)
Model C (ft GTSDb)	0.993	0.99 _(0.656)	0.992	0.97 _(0.749)

Taula 12. Avaluació de cada un dels models per a cada un dels conjunts de dades.

Anteriorment, hem vist que per visualitzar el rendiment dels models hem usat les gràfiques *precision-recall*. Amb aquestes gràfiques obtenim la mètrica *mean Average Precision* (mAP@0.5). El valor 0.5 que es mostra davant de l'@ és el llindar de confiança (*threshold*), tal com hem explicat a l'apartat de mètriques.

Aquest llindar ens indica que agafarem com a deteccions aquelles amb confiança major o igual a 0.5. Per aquelles amb menor confiança, tot i ser detectades, no es consideraran deteccions. Per la mètrica de *F1 score* trobem un llindar, aquesta vegada proporcionat pel model. Aquest serà el llindar òptim de separació entre deteccions correctes i incorrectes.

Al final del punt 8.4, he comparat els senyals dels dos conjunts, i he trobat que pràcticament no hi ha diferència entre ells.

Observant els resultats de la taula 12, trobem que el rendiment de cada model evoluciona ascendentment pels dos conjunts, menys en el cas del model B testejat amb GTSDb. Es pot entendre, perquè el model B no s'ha entrenat en cap moment amb GTSDb.

Podem veure que el rendiment del model C és més elevat amb EuRoSign que amb GTSDb. Per tant, trobem que un model entrenat primer amb senyals europeus i després amb senyals d'Alemanya ofereix un millor rendiment pel reconeixement de senyals europeus, que si només s'hagués entrenat amb senyals europees.

Ara bé, no podem comparar ambdós entrenaments perquè un s'ha realitzat amb 20 èpoques i l'altre amb 100, i si a més els senyals entre conjunts són tan semblants, és lògic que un model amb un major entrenament tingui més bon rendiment independentment del conjunt amb el qual s'avalua.

En definitiva, el rendiment del nostre sistema de detecció i classificació serà directament proporcional al temps d'entrenament d'aquest per a qualsevol conjunt de validació, a causa de la semblança entre els senyals d'ambdós conjunts.

8.7. Inferència sobre video en temps real

Per concloure, he volgut provar un model en una inferència sobre un vídeo, per observar el seu comportament en una aplicació que simuli la inferència en temps real. Per a aquesta prova, hem utilitzat el model C, que ha demostrat ser el més rendible segons els resultats obtinguts.

L'enllaç a la inferència es pot consultar [aquí](#). El codi de la inferència el podem trobar a l'annex del treball.

Els resultats obtinguts són coherents amb les prediccions realitzades sobre imatges, demostrant així la consistència del model en diferents aplicacions.

9. Conclusions del treball

9.1. Problemes sorgits

9.1.1. CPU i GPU

Inicialment, volia executar el codi per realitzar l'entrenament al meu ordinador. Però després de documentar-me sobre les prestacions de la CPU i GPU del meu ordinador portàtil, vaig preferir executar el codi al núvol.

Durant el grau vaig descobrir l'entorn Google Colaboratory i ja coneixia les múltiples opcions que ofereix per a l'execució al núvol fent ús de les seves CPUs, GPUs i TPUs.

A priori havia de realitzar l'entrenament amb la TPU que és la tecnologia específica pel *deep learning* i no la targeta gràfica GPU. A les següents figures 76 i 77 mostro el temps d'entrenament de les tres primeres èpoques d'un entrenament en igualtat de condicions per la GPU (T4 GPU) i TPU (v2-8 TPU).

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
1/100	3.76G	0.6951	0.4229	0.809	18	640: 100% 90/90 [00:29<00:00, 3.03it/s]
	Class	Images	Instances	Box(P	R	mAP50 mAP50-95): 100% 12/12 [00:04<00:00, 2.69it/s]
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
2/100	3.73G	0.8016	0.5413	0.8242	12	640: 100% 90/90 [00:28<00:00, 3.12it/s]
	Class	Images	Instances	Box(P	R	mAP50 mAP50-95): 100% 12/12 [00:03<00:00, 3.24it/s]
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
3/100	3.71G	0.8444	0.5376	0.8308	15	640: 100% 90/90 [00:27<00:00, 3.28it/s]
	Class	Images	Instances	Box(P	R	mAP50 mAP50-95): 100% 12/12 [00:02<00:00, 4.42it/s]
	all	180	230	0.927	0.933	0.958 0.754

Figura 76. Entrenament realitzat amb GPU

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
1/3	0G	0.6956	0.4094	0.8126	13	640: 100% 90/90 [12:53<00:00, 8.59s/it]
	Class	Images	Instances	Box(P	R	mAP50 mAP50-95): 100% 12/12 [00:37<00:00, 3.10s/it]
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
2/3	0G	0.7389	0.454	0.8111	11	640: 100% 90/90 [12:05<00:00, 8.06s/it]
	Class	Images	Instances	Box(P	R	mAP50 mAP50-95): 100% 12/12 [00:37<00:00, 3.09s/it]
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
3/3	0G	0.6815	0.4242	0.8013	11	640: 100% 90/90 [12:02<00:00, 8.03s/it]
	Class	Images	Instances	Box(P	R	mAP50 mAP50-95): 100% 12/12 [00:38<00:00, 3.20s/it]

Figura 77. Entrenament realitzat amb TPU

Podem veure com el temps d'una època amb GPU és ≈ 28 segons mentre que amb la TPU és ≈ 12 minuts.

He trobat que la TPU normalment treballa amb *datasets* de més imatges i, per tant, utilitza una mida de *batch* més gran. Per aquest *batch* el temps d'entrenament del model es redueix.

L'experiment A el baix realitzar amb la TPU, però al veure el temps elevat que trigava en entrenar-se, vaig preferir fer servir la GPU pels altres experiments.

9.1.2. Avaluació de diferents classes

La idea inicial del projecte era avaluar cada un dels models amb EuRoSign i GTSDb. Quan vaig voler avaluar el model A (entrenat amb el *dataset* COCO) amb el dataset EuRoSign no ho vaig poder fer. No havia caigut que un model entrenat amb un *dataset* de 80 classes (COCO), no es pot avaluar amb un de 4 classes, independentment de si les classes dels diferents conjunts tenen o no relació.

La solució que se'm va ocórrer va ser fer un *fine-tuning* d'una sola època del model preentrenat de COCO amb el conjunt EuRoSign. D'aquesta manera, sí que vaig poder avaluar el model. Un model preentrenat amb COCO amb molt poc aprenentatge sobre el conjunt EuRoSign.

9.1.3. Conjunts de dades similars

Segons les directrius establertes pel *Conveni de Viena sobre Senyalització Viària (1968)*, s'han exposat unes normes estàndard per la perfecció dels senyals de trànsit, amb l'objectiu de millorar la seguretat i facilitar de comprensió per part de conductors d'arreu del món.

Els senyals europeus es van dissenyar seguint aquest estàndard. El que es buscava era precisament tenir una semblança en forma, disseny i color, entre els senyals d'una mateixa regió.

Per aquest motiu, es podria haver considerat treballar amb dos conjunts de dades més diversos, amb senyals provinents de continents diferents, amb l'objectiu d'aportar una major riquesa i varietat als resultats obtinguts.

9.2. Conclusions

En aquest treball, hem explorat la implementació d'un sistema de detecció i classificació de senyals de trànsit utilitzant models basats en YOLO i dades d'entrenament provinents de conjunts específics com EuRoSign i GTSDb. Els resultats obtinguts mostren que el rendiment del sistema està condicionat tant per la qualitat del conjunt de dades, com per la quantitat d'entrenament del model.

Hem constatat que un entrenament més llarg i específic pot millorar el rendiment del model, però també pot introduir el risc de *catastrophic forgetting* si no es gestiona correctament. Tot i això, els nostres resultats qualitius mostren que les similituds entre els conjunts EuRoSign i GTSDb minimitzen aquest efecte.

Així mateix, hem trobat que el rendiment global del model millora gràcies el *fine-tuning* progressiu sobre els conjunts de dades cada vegada més especialitzats.

Aquest estudi reivindica la importància d'una estratègia adequada per a l'entrenament de models en tasques de visió artificial, especialment en contextos crítics com els sistemes de conducció autònoma, on la precisió i la rapidesa en la detecció són imprescindibles per garantir la seguretat viària.

9.3. Futures línies d'investigació

L'objectiu principal en aquesta àrea recau en la millora contínua dels models de detecció i classificació per obtenir prediccions més precises i fiables, disminuint la velocitat.

Per assolir aquests objectius, es requereix un coneixement profund en xarxes neuronals, tant pel que fa a la seva estructura com als mecanismes d'entrenament, ja que la seva optimització interna pot marcar una diferència significativa en el rendiment del sistema.

Es podria explorar la combinació de xarxes convolucionals i *transformers*, com el *Vision Transformer*, que ens permeten aprofitar la capacitat de les CNNs per extreure característiques locals i la dels transformers per capturar relacions complexes entre objectes dins d'una imatge. Aquesta fusió és molt explorada darrerament, donant lloc a models extraordinaris en termes de precisió.

10. Bibliografía

- [1] Wu, Wei. 14 April 2024. "Unsupervised Learning" (PDF). https://na.uni-tuebingen.de/ex/ml_seminar_ss2022/Unsupervised_Learning%20Final.pdf [Consulta; Setembre, 2024.]
- [2] Liu, Xiao; Zhang, Fanjin; Hou, Zhenyu; Mian, Li; Wang, Zhaoyu; Zhang, Jing; Tang, Jie (2021). "Self-supervised Learning: Generative or Contrastive". *IEEE Transactions on Knowledge and Data Engineering*: 1. arXiv:2006.08218. doi:[10.1109/TKDE.2021.3090866](https://doi.org/10.1109/TKDE.2021.3090866) [Consulta; Setembre, 2024.]
- [3] Radford, Alec; Narasimhan, Karthik; Salimans, Tim; Sutskever, Ilya (11 June 2018). "Improving Language Understanding by Generative Pre-Training" (PDF). OpenAI. p. 12. Archived (PDF) from the original on 26 January 2021. Retrieved 23 January 2021. [Consulta; Setembre, 2024.]
- [4] Gorini, M. (2024, 25 octubre). ¿Cuál es la diferencia entre el machine learning y el deep learning? *BiSmart*. <https://blog.bismart.com/diferencia-machine-learning-deep-learning> [Consulta; Setembre, 2024.]
- [5] Rojewska, K. (2024, 8 marzo). *What are Neural Networks and What are Their Applications?* Qtravel.ai. <https://www.qtravel.ai/blog/what-are-neural-networks-and-what-are-their-applications/> [Consulta; Setembre, 2024.]
- [6] Saturn Cloud Blog. (2023, 20 noviembre). *A Guide to Convolutional Neural Networks — the ELI5 way* <https://saturncloud.io/blog/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way/> [Consulta; Setembre, 2024.]
- [7] IBM. (2024, 21 agosto). *redes neuronales convolucionales*. IBM. <https://www.ibm.com/es-es/topics/convolutional-neural-networks> [Consulta; Octubre, 2024.]
- [8] Steen, D. (2021, 16 diciembre). Precision-Recall Curves - Doug Steen - medium. *Medium*. <https://medium.com/@douglassteen/precision-recall-curves-d32e5b290248> [Consulta; Octubre, 2024.]

- [9] Boesch, G. (2024, 18 septiembre). *Computer Vision Model Performance Evaluation (Guide 2024.)* viso.ai. <https://viso.ai/computer-vision/model-performance/>
[Consulta; Octubre, 2024.]
- [10] LinkeDIN. (2023, 16 agosto). *How do you assess computer vision model performance?* <https://www.linkedin.com/advice/3/how-do-you-assess-computer-vision-model>
[Consulta; Octubre, 2024.]
- [11] Tomar, N. (2023, 9 mayo). What is Intersection over Union (IoU) in Object Detection? — Idiot Developer. *Medium*.
<https://nikhilroxtomar.medium.com/what-is-intersection-over-union-iou-in-object-detection-idiot-developer-39448ac56431>
[Consulta; Octubre, 2024.]
- [12] Pavnov, H. (2022, 7 mayo). Mean Average Precision (mAP) Explained: Everything You Need to Know. V7.
<https://www.v7labs.com/blog/mean-average-precision>
[Consulta; Octubre, 2024.]
- [13] *What Makes a Good Dataset?* (s. f.).
<https://docs.appian.com/suite/help/24.3/collect-data.html>
[Consulta; Octubre, 2024.]
- [14] *Public Archive: ff17dc924eba88d5d01a807357d6614c.* (s. f.).
<https://erda.ku.dk/public/archives/ff17dc924eba88d5d01a807357d6614c/published-archive.tml>
[Consulta; Octubre, 2024.]
- [15] *Traffic Signs Dataset in YOLO format.* (2020, 3 abril). Kaggle.
<https://www.kaggle.com/datasets/valentynsichkar/traffic-signs-dataset-in-yolo-format>
[Consulta; Octubre, 2024.]
- [16] Chakure, A. (2024, 10 agosto). YOLOv3 (You Only Look Once) Deep Learning Algorithm. All you need to know.. | by Afroz Chakure | Apr, 2021 | Medium | DataDrivenInvestor. *Medium*.
<https://medium.datadriveninvestor.com/yolov3-you-only-look-once-12de76ad74d5>
[Consulta; Octubre, 2024.]
- [17] *Traffic Signs Dataset in YOLO format.* (2020b, abril 3). Kaggle.
<https://www.kaggle.com/datasets/valentynsichkar/traffic-signs-dataset-in-yolo-format>
[Consulta; Noviembre, 2024.]

[18] Ankitp. (2021, 8 agosto). *Step by step YOLOv3-Object detection*. Kaggle. <https://www.kaggle.com/code/ankitp013/step-by-step-yolov3-object-detection/notebook>

[Consulta; Desembre, 2024.]

[19] Shaunak Halbe. A simple explanation to the YOLO Algorithm -, 111803053 , Division 1 T-2 Batch

[Consulta; Desembre, 2024.]

[20] Wickramasinghe, Y. S. (2025, 6 enero). *Best Object Detection Models (2024 list)*. DagsHub Blog. <https://dagshub.com/blog/best-object-detection-models/>

[Consulta; Desembre, 2024.]

[21] GeeksforGeeks. (2024, 19 abril). *10 Best YOLO (You Only Look Once) Alternatives for RealTime Object Detection in 2024*. GeeksforGeeks.

<https://www.geeksforgeeks.org/yolo-you-only-look-once-alternatives/>

[Consulta; Gener, 2025.]

[22] X. Liu et al., "Self-Supervised Learning: Generative or Contrastive," in IEEE Transactions on Knowledge and Data Engineering, vol. 35, no. 1, pp. 857-876, 1 Jan. 2023 <https://ieeexplore.ieee.org/document/9462394>

[Consulta; Gener, 2025.]

[23] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. In

Advances in neural information processing systems, pages 153–160, 2007.

https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf

[Consulta; Gener, 2025.]

11. Annex

11.1. Notebook de preparació pel conjunt EuRoSign

Aquest *notebook* és l'encarregat de preparar el conjunt de dades EuRoSign.

```
from google.colab import drive

# Connectem google drive amb colab
drive.mount('/content/drive')

import os
import shutil
import random
import cv2
from matplotlib import pyplot as plt
from PIL import Image
from sklearn.model_selection import train_test_split

source_folder = '/content/drive/MyDrive/TFG/YOLO/dataset1/ts/ts'
destination_folder = '/content/drive/MyDrive/TFG/YOLO/EuRoSign_prepared'

# Creem les subcarpetes de destinació
images_destination = os.path.join(destination_folder, 'images')
labels_destination = os.path.join(destination_folder, 'labels')

os.makedirs(images_destination, exist_ok=True)
os.makedirs(labels_destination, exist_ok=True)

# Processem cada fitxer, tant els .txt com .jpg
for file_name in os.listdir(source_folder):
    source_path = os.path.join(source_folder, file_name)

    # Comprovem si és una imatge o un fitxer d'etiquetes
    if file_name.endswith('.jpg'):
        destination_path = os.path.join(images_destination, file_name)
        shutil.copy(source_path, destination_path)
    elif file_name.endswith('.txt'):
        destination_path = os.path.join(labels_destination, file_name)
        shutil.copy(source_path, destination_path)

# Comprovem la quantitat de fitxers al nou directori
prepared_folder = '/content/drive/MyDrive/TFG/YOLO/EuRoSign_prepared'

images_folder = os.path.join(prepared_folder, 'images')
labels_folder = os.path.join(prepared_folder, 'labels')

num_images = len([f for f in os.listdir(images_folder) if os.path.isfile(os.path.join(images_folder, f)) and f.endswith('.jpg')])
num_labels = len([f for f in os.listdir(labels_folder) if os.path.isfile(os.path.join(labels_folder, f)) and f.endswith('.txt')])

print(f"Nombre total d'imatges: {num_images}")
print(f"Nombre total d'etiquetes: {num_labels}")
```



```

# Script per mostrar alguns exemples etiquetats del conjunt
images_folder = '/content/drive/MyDrive/TFG/YOLO/EuRoSign_prepared/images/'
labels_folder = '/content/drive/MyDrive/TFG/YOLO/EuRoSign_prepared/labels/'

image_files = sorted([f for f in os.listdir(images_folder) if os.path.isfile(os.path.join(images_folder, f))])

# Seleccionem aleatòriament 6 imatges
selected_images = random.sample(image_files, 6)

def read_labels(label_file_path):
    with open(label_file_path, 'r') as f:
        lines = f.readlines()
    return [line.strip() for line in lines]

fig, axes = plt.subplots(3, 2, figsize=(12, 18))

for i, image_name in enumerate(selected_images):
    image_path = os.path.join(images_folder, image_name)
    img = cv2.imread(image_path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    label_file_path = os.path.join(labels_folder, f"{os.path.splitext(image_name)[0]}.txt")
    labels = read_labels(label_file_path)
    labels_text = "\n".join(labels) if labels else "No labels"

    ax = axes[i // 2, i % 2]
    ax.imshow(img)
    ax.set_title(f"{image_name}\n{labels_text}", fontsize=10)
    ax.axis('off')

plt.subplots_adjust(wspace=0.5, hspace=0)
plt.show()

```

```

# Script per dividir el conjunt entre el subconjunt d'entrenament i validació usant sk.learn
images_folder = '/content/drive/MyDrive/TFG/YOLO/EuRoSign_prepared/images'
labels_folder = '/content/drive/MyDrive/TFG/YOLO/EuRoSign_prepared/labels'

base_dest_dir = '/content/drive/MyDrive/TFG/YOLO/EuRoSign_splited'

dest_dirs = {
    'train': {'images': os.path.join(base_dest_dir, 'train/images'),
            'labels': os.path.join(base_dest_dir, 'train/labels')},
    'val': {'images': os.path.join(base_dest_dir, 'val/images'),
            'labels': os.path.join(base_dest_dir, 'val/labels')}
}

for split, paths in dest_dirs.items():
    for path in paths.values():
        os.makedirs(path, exist_ok=True)

image_files = sorted([f for f in os.listdir(images_folder) if f.endswith('.jpg')])
label_files = sorted([f for f in os.listdir(labels_folder) if f.endswith('.txt')])

common_files = [f for f in image_files if f.replace('.jpg', '.txt') in label_files]

# Proporció entre train i val (80%-20%)
train_files, val_files = train_test_split(common_files, test_size=0.2, random_state=42)

# Movem els fitxers
def copy_files(file_list, split_name):
    for file_name in file_list:
        shutil.copy(os.path.join(images_folder, file_name),
                    os.path.join(dest_dirs[split_name]['images'], file_name))

        label_name = file_name.replace('.jpg', '.txt')
        shutil.copy(os.path.join(labels_folder, label_name),
                    os.path.join(dest_dirs[split_name]['labels'], label_name))

copy_files(train_files, 'train')
copy_files(val_files, 'val')

```

11.2. Notebook de preparació pel conjunt GTSDb

Aquest *notebook* és l'encarregat de preparar el conjunt de dades de GTSDb..

```

from google.colab import drive

# Connectem google drive amb colab
drive.mount('/content/drive')

!pip install opencv-python

import os
import cv2
import random
from matplotlib import pyplot as plt
from PIL import Image
import shutil
from sklearn.model_selection import train_test_split

```

```

# INPUT
dataset_folder = '/content/drive/MyDrive/TFG/YOLO/GTSDb'
images_folder = os.path.join(dataset_folder, 'images')

gt_file = os.path.join(dataset_folder, 'gt.txt')

# OUTPUT
drive_output_folder = '/content/drive/MyDrive/TFG/YOLO/GTSDb_prepared'

# Creem les carpetes output
os.makedirs(os.path.join(drive_output_folder, 'images'), exist_ok=True)
os.makedirs(os.path.join(drive_output_folder, 'labels'), exist_ok=True)
images_folder_output = os.path.join(drive_output_folder, 'images')
labels_folder_output = os.path.join(drive_output_folder, 'labels')

all_image_files = {f for f in os.listdir(images_folder) if os.path.isfile(os.path.join(images_folder, f))}

# Llegim el fitxer gt.txt
with open(gt_file, 'r') as f:
    lines = f.readlines()

# Associem les 43 classes en 4
def map_class_id(original_id):
    prohibitory = {0, 1, 2, 3, 4, 5, 7, 8, 9, 10, 15, 16}
    danger = {11, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31}
    mandatory = {33, 34, 35, 36, 37, 38, 39, 40}
    other = {6, 12, 13, 14, 17, 32, 41, 42}

    if original_id in prohibitory:
        return 0
    elif original_id in danger:
        return 1
    elif original_id in mandatory:
        return 2
    elif original_id in other:
        return 3
    else:
        raise ValueError(f"Classe desconeguda: {original_id}")

```

```

# Processem les anotacions
annotations = {}
for line in lines:
    elements = line.strip().split(';')
    image_name, left, top, right, bottom, class_id = elements
    left, top, right, bottom, class_id = map(int, [left, top, right, bottom, class_id])

    mapped_class_id = map_class_id(class_id)

    if image_name not in annotations:
        annotations[image_name] = []
    annotations[image_name].append((left, top, right, bottom, mapped_class_id))

for image_name in all_image_files:
    image_path = os.path.join(images_folder, image_name)

    label_file_path = os.path.join(labels_folder_output, f"{os.path.splitext(image_name)[0]}.txt")
    output_image_path = os.path.join(images_folder_output, f"{os.path.splitext(image_name)[0]}.jpg")

    img = cv2.imread(image_path)
    if img is None:
        print(f"Error llegint la imatge: {image_name}")
        continue
    height, width, _ = img.shape

    if image_name in annotations:
        with open(label_file_path, 'w') as label_file:
            for left, top, right, bottom, class_id in annotations[image_name]:
                # Calculem el centre de la bounding box
                x_center = ((left + right) / 2) / width
                y_center = ((top + bottom) / 2) / height
                bbox_width = (right - left) / width
                bbox_height = (bottom - top) / height

                # Escrivim l'etiqueta
                label_file.write(f"{class_id} {x_center:.6f} {y_center:.6f} {bbox_width:.6f} {bbox_height:.6f}\n")
    else:
        open(label_file_path, 'w').close()

    # Desem la imatge
    cv2.imwrite(output_image_path, img)

```

```

# Comptem quants fitxers hi ha a la carpeta
folder_path_images = '/content/drive/MyDrive/TFG/YOLO/GTSDB_prepared/images'
folder_path_labels = '/content/drive/MyDrive/TFG/YOLO/GTSDB_prepared/labels'

file_count_images = len([f for f in os.listdir(folder_path_images) if os.path.isfile(os.path.join(folder_path_images, f))])
file_count_labels = len([f for f in os.listdir(folder_path_labels) if os.path.isfile(os.path.join(folder_path_labels, f))])

assert file_count_images == 900, f"No s'han guardat totes les imatges."
assert file_count_labels == 900, f"No s'han guardat totes les etiquetes."

```

```

# Script per mostrar alguns exemples etiquetats del conjunt
images_folder = '/content/drive/MyDrive/TFG/YOLO/GTSDB_prepared/images'
labels_folder = '/content/drive/MyDrive/TFG/YOLO/GTSDB_prepared/labels'

image_files = sorted([f for f in os.listdir(images_folder) if os.path.isfile(os.path.join(images_folder, f))])

selected_images = random.sample(image_files, 9)

def read_labels(label_file_path):
    with open(label_file_path, 'r') as f:
        lines = f.readlines()
    return [line.strip() for line in lines]

fig, axes = plt.subplots(3, 3, figsize=(12, 12))

for i, image_name in enumerate(selected_images):
    image_path = os.path.join(images_folder, image_name)
    img = cv2.imread(image_path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    label_file_path = os.path.join(labels_folder, f"{os.path.splitext(image_name)[0]}.txt")
    labels = read_labels(label_file_path)
    labels_text = "\n".join(labels) if labels else "No labels"

    ax = axes[i // 3, i % 3]
    ax.imshow(img)
    ax.set_title(f"{image_name}\n{labels_text}")
    ax.axis('off')

plt.subplots_adjust(wspace=0.5, hspace=0.4)
plt.tight_layout()
plt.show()

```

```

# Script per dividir el conjunt entre el subconjunt d'entrenament i validació usant sk.learn
images_folder = '/content/drive/MyDrive/TFG/YOLO/GTSDB_prepared/images'
labels_folder = '/content/drive/MyDrive/TFG/YOLO/GTSDB_prepared/labels'

base_dest_dir = '/content/drive/MyDrive/TFG/YOLO/GTSDB_splited'

dest_dirs = {
    'train': {'images': os.path.join(base_dest_dir, 'train/images'),
            'labels': os.path.join(base_dest_dir, 'train/labels')},
    'val': {'images': os.path.join(base_dest_dir, 'val/images'),
           'labels': os.path.join(base_dest_dir, 'val/labels')}
}

for split, paths in dest_dirs.items():
    for path in paths.values():
        os.makedirs(path, exist_ok=True)

image_files = sorted([f for f in os.listdir(images_folder) if f.endswith('.jpg')])
label_files = sorted([f for f in os.listdir(labels_folder) if f.endswith('.txt')])

common_files = [f for f in image_files if f.replace('.jpg', '.txt') in label_files]

# Proporció entre train i val (80%-20%)
train_files, val_files = train_test_split(common_files, test_size=0.2, random_state=42)

# Movem els fitxers
def copy_files(file_list, split_name):
    for file_name in file_list:
        shutil.copy(os.path.join(images_folder, file_name),
                    os.path.join(dest_dirs[split_name]['images'], file_name))

        label_name = file_name.replace('.jpg', '.txt')
        shutil.copy(os.path.join(labels_folder, label_name),
                    os.path.join(dest_dirs[split_name]['labels'], label_name))

copy_files(train_files, 'train')
copy_files(val_files, 'val')

```

11.3. *Notebook* de treball dels models

Aquest *notebook* és l'encarregat de realitzar els entrenaments i l'avaluació dels models del treball, entre d'altres coses.

```

from google.colab import drive

# Connectem google drive amb colab
drive.mount('/content/drive')

```

```

!pip install ultralytics
!pip install pillow
!pip install numpy

```

```

import os
import random
from PIL import Image
import shutil
from ultralytics import YOLO

```



```

# Funció per importar les dades dels conjunts EuRoSign
def import_data(source_dir, destination_dir):
    if os.path.exists(destination_dir):
        print(f"La carpeta {destination_dir} ja existeix.")
        return

    shutil.copytree(source_dir, destination_dir)
    print(f"Carpeta copiada {source_dir} a {destination_dir}")

# Comprovem que les dades s'hagin importat correctament
def count_images_and_labels(image_folder, label_folder, extension_image=".jpg", extension_label=".txt"):
    image_files = [f for f in os.listdir(image_folder) if f.endswith(extension_image)]
    label_files = [f for f in os.listdir(label_folder) if f.endswith(extension_label)]

    matching_files = [f for f in image_files if f.replace(extension_image, extension_label) in label_files]
    return len(matching_files)

train_images_folder = os.path.join(destination_dir, 'train/images')
train_labels_folder = os.path.join(destination_dir, 'train/labels')
val_images_folder = os.path.join(destination_dir, 'val/images')
val_labels_folder = os.path.join(destination_dir, 'val/labels')

if os.path.exists(train_images_folder) and os.path.exists(train_labels_folder) and \
    os.path.exists(val_images_folder) and os.path.exists(val_labels_folder):

    train_count = count_images_and_labels(train_images_folder, train_labels_folder)
    val_count = count_images_and_labels(val_images_folder, val_labels_folder)

    print(f"Hi ha {train_count} imatges/etiquetes a la carpeta 'train'.")
    print(f"Hi ha {val_count} imatges/etiquetes a la carpeta 'val'.")

# Importem EuRoSign
source_dir = '/content/drive/MyDrive/TFG/YOLO/EuRoSign_split/'
destination_dir = '/content/EuRoSign/'
import_data(source_dir, destination_dir)

# Importem GTSDb
source_dir = '/content/drive/MyDrive/TFG/YOLO/GTSDb_split/'
destination_dir = '/content/GTSDb/'
import_data(source_dir, destination_dir)

```

```

# Creem un fitxer de configuració YAML per al dataset EuRoSign
%%writefile /content/EuRoSign.yaml

```

```

train: /content/EuRoSign/train/A
val: /content/EuRoSign/val/

nc: 4
names: ['prohibitory', 'danger', 'mandatory', 'other']

```

```

# Creem un fitxer de configuració YAML per al dataset GTSDb
%%writefile /content/GTSDb.yaml

```

```

train: /content/GTSDb/train/
val: /content/GTSDb/val/

nc: 4
names: ['prohibitory', 'danger', 'mandatory', 'other']

```

```
# Entrenament per obtenir el model A partir d'un model preentrenat de COCO (yolov8m.pt)
model = YOLO('yolov8m.pt')

A_train = model.train(
    data='/content/EuRoSign.yaml',
    epochs=1,
    imgsz=[1360, 800],
    batch=1,
    workers=8,
    project="/content/drive/MyDrive/TFG/YOLO/resultats/",
    name='A-train',
    save_json=True)

print(A_train)
```

```
# Entrenament per obtenir el model B (fine-tuning amb el conjunt EuRoSign)
from ultralytics import YOLO
model = YOLO('/content/drive/MyDrive/TFG/YOLO/resultats/A-train/weights/best.pt')

B_train = model.train(
    data='/content/EuRoSign.yaml',
    epochs=20,
    imgsz=640,
    batch=2, # Redueix el batch size (comença amb 8 i ajusta si cal)
    project="/content/drive/MyDrive/TFG/YOLO/resultats/",
    name='B-train'
)

print(B_train)
```

```
# Entrenament per obtenir el model C (fine-tuning amb el conjunt GTSDB)
model = YOLO('/content/drive/MyDrive/TFG/YOLO/resultats/B-train/weights/best.pt')

C_train = model.train(
    data='/content/GTSDB.yaml',
    epochs=100,
    imgsz=640,
    batch=8,
    project="/content/drive/MyDrive/TFG/YOLO/resultats/",
    name='C-train',
    save_json=True)

print(C_train)
```

```
# Entrenament addicional per comparar el temps d'entrenament entre GPU i TPU
model = YOLO('/content/drive/MyDrive/TFG/YOLO/resultats/B-train/weights/best.pt')

C_train = model.train(
    data='/content/GTSDB.yaml',
    epochs=3,
    imgsz=640,
    batch=8,
    project="/content/drive/MyDrive/TFG/YOLO/resultats/",
    name='TPU-C-train',
    save_json=True)

print(C_train)
```

```
# Validació MODEL A per EuRoSign
model = YOLO('/content/drive/MyDrive/TFG/YOLO/resultats/A-train/weights/best.pt')

val_A_EuRoSign = model.val(
    data='/content/EuRoSign.yaml',
    split='val',
    project='/content/drive/MyDrive/TFG/YOLO/resultats/',
    name='val-A-EuRoSign',
    save=True,
    save_json=True,
    plots=True,
    verbose=True,
    save_crop=True
)

print(val_A_EuRoSign)
```

```
# Validació MODEL A per GTSDB
model = YOLO('/content/drive/MyDrive/TFG/YOLO/resultats/A-train/weights/best.pt')

val_A_GTSDB = model.val(
    data='/content/GTSDB.yaml',
    split='val',
    project='/content/drive/MyDrive/TFG/YOLO/resultats/',
    name='val-A-GTSDB',
    save=True,
    save_json=True,
    plots=True,
    verbose=True,
    save_crop=True
)

print(val_A_GTSDB)
```

```
# Validació MODEL B per EuRoSign
model = YOLO('/content/drive/MyDrive/TFG/YOLO/resultats/B-train/weights/best.pt')

val_B_EuRoSign = model.val(
    data='/content/EuRoSign.yaml',
    split='val',
    imgsz=[1360, 800],
    project='/content/drive/MyDrive/TFG/YOLO/resultats/',
    name='val-B-EuRoSign',
    save=True,
    save_json=True,
    plots=True,
    verbose=True,
    save_crop=True
)

print(val_B_EuRoSign)
```

```
# Validació MODEL B per GTSDB
model = YOLO('/content/drive/MyDrive/TFG/YOLO/resultats/B-train/weights/best.pt')

val_B_GTSDB = model.val(
    data='/content/GTSDB.yaml',
    split='val',
    imgsz=[1360, 800],
    project='/content/drive/MyDrive/TFG/YOLO/resultats/',
    name='val-B-GTSDB',
    save=True,
    save_json=True,
    plots=True,
    verbose=True,
    save_crop=True
)

print(val_B_GTSDB)
```

```

# Validació MODEL C per EuRoSign
model = YOLO('/content/drive/MyDrive/TFG/YOLO/resultats/C-train/weights/best.pt')

val_C_EuRoSign = model.val(
    data='/content/EuRoSign.yaml',
    split='val',
    project='/content/drive/MyDrive/TFG/YOLO/resultats/',
    name='val-C-EuRoSign',
    save=True,
    save_json=True,
    plots=True,
    verbose=True,
    save_crop=True
)

print(val_C_EuRoSign)

```

```

# Validació MODEL C per GTSDB
model = YOLO('/content/drive/MyDrive/TFG/YOLO/resultats/C-train/weights/best.pt')

val_C_GTSDB = model.val(
    data='/content/GTSDB.yaml',
    split='val',
    project='/content/drive/MyDrive/TFG/YOLO/resultats/',
    name='val-C-GTSDB',
    save=True,
    save_json=True,
    plots=True,
    verbose=True,
    save_crop=True
)

print(val_C_GTSDB)

```

```

import cv2
from PIL import Image
from google.colab.patches import cv2_imshow

# Script per realitzar inferència sobre video
def infer_video(video_path, output_path, model_path):
    model = YOLO(model_path)

    cap = cv2.VideoCapture(video_path)
    if not cap.isOpened():
        print("Error: No s'ha pogut obrir el vídeo.")
        return

    width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    fps = int(cap.get(cv2.CAP_PROP_FPS))

    fourcc = cv2.VideoWriter_fourcc(*'mp4v')
    out = cv2.VideoWriter(output_path, fourcc, fps, (width, height))

    while True:
        ret, frame = cap.read()
        if not ret:
            break

        results = model.predict(frame, imgsz=640)
        annotated_frame = results[0].plot()
        out.write(annotated_frame)
        cv2_imshow(annotated_frame)

    cap.release()
    out.release()

video_path = "/content/drive/MyDrive/TFG/tfg_inference_video.mp4"
output_path = "/content/drive/MyDrive/TFG/YOLO/resultats/video_inference2.mp4"
model_path = "/content/drive/MyDrive/TFG/YOLO/resultats/C-train/weights/best.pt"
infer_video(video_path, output_path, model_path)

```

```

# Script per conèixer el tamany de les imatges d'un dataset
def get_image_size(image_path):
    with Image.open(image_path) as img:
        return img.size

# Funció per obtenir totes les mides úniques de les imatges dins d'un directori
def get_unique_image_sizes(directory_path):
    unique_sizes = set()

    for filename in os.listdir(directory_path):
        file_path = os.path.join(directory_path, filename)

        if os.path.isfile(file_path) and filename.lower().endswith(('.jpg', '.jpeg', '.png', '.bmp')):
            try:
                size = get_image_size(file_path)
                unique_sizes.add(size)
            except Exception as e:
                print(f"No s'ha pogut processar {filename}: {e}")

    return unique_sizes

directory_path = '/content/GTSDDB/train/images'
unique_sizes = get_unique_image_sizes(directory_path)

print("Mides úniques:")
for size in unique_sizes:
    print(f"{size[0]} x {size[1]}")

```



```

def count_detections_by_class(paths):
    class_counts = {0: 0, 1: 0, 2: 0, 3: 0}

    for path in paths:
        if not os.path.exists(path):
            print(f"La ruta {path} no existeix.")
            continue

        for filename in os.listdir(path):
            if filename.endswith('.txt'):
                filepath = os.path.join(path, filename)
                with open(filepath, 'r') as file:
                    for line in file:
                        parts = line.strip().split()
                        if parts:
                            class_id = int(parts[0])
                            if class_id in class_counts:
                                class_counts[class_id] += 1

    return class_counts

def plot_class_distribution(class_counts, dataset_name="Dataset"):
    labels = ['Prohibitory', 'Danger', 'Mandatory', 'Other']
    counts = [class_counts[i] for i in range(4)]
    total = sum(counts)

    if total == 0:
        print(f"No s'han trobat deteccions en el dataset {dataset_name}.")
        return

    percentages = [(count / total) * 100 for count in counts]

    fig, ax = plt.subplots(figsize=(8, 6))

    bars = ax.bar(labels, counts, color=['red', 'orange', 'blue', 'green'])
    ax.set_title(f'Distribució de deteccions per classe - {dataset_name}')
    ax.set_ylabel('Nombre de deteccions')

    for bar, count, percentage in zip(bars, counts, percentages):
        ax.text(bar.get_x() + bar.get_width() / 2, bar.get_height() + 1,
                f'{count}\n({percentage:.1f}%)',
                ha='center', va='bottom')

    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.show()

paths_EuRoSign = [
    '/content/GTSDb/val/labels'',
]

class_counts_EuRoSign = count_detections_by_class(paths_EuRoSign)
plot_class_distribution(class_counts_EuRoSign, dataset_name="EuRoSign")

```

```

# Script preparat per mostrar les bounding box predites sobre una imatge

def draw_bounding_boxes(image_path, json_path, score_threshold=0.5):
    image = cv2.imread(image_path)
    if image is None:
        print(f"Error: No s'ha pogut carregar la imatge de la ruta {image_path}.")
        return

    image_id = int(os.path.splitext(os.path.basename(image_path))[0])

    try:
        with open(json_path, 'r') as f:
            data = json.load(f)
    except FileNotFoundError:
        print(f"Error: No s'ha trobat el fitxer JSON a la ruta {json_path}.")
        return
    except json.JSONDecodeError:
        print("Error: El fitxer JSON no és vàlid.")
        return

    category_colors = {
        1: (0, 0, 255),    # Vermell (BGR)
        2: (0, 255, 255),  # Groc (BGR)
        3: (255, 0, 0),    # Blau (BGR)
        4: (255, 255, 255) # Blanc (BGR)
    }

    category_labels = {
        1: "PROHIBITORY",
        2: "DANGER",
        3: "MANDATORY",
        4: "OTHER"
    }

    filtered_data = [obj for obj in data if obj['image_id'] == image_id and round(obj['score'], 1) >= score_threshold]

    # Dibuix de les bounding boxes
    for obj in filtered_data:
        bbox = obj['bbox']
        category_id = obj['category_id']
        score = round(obj['score'], 1)

        x, y, w, h = map(int, bbox)

        color = category_colors.get(category_id, (255, 255, 255))
        cv2.rectangle(image, (x, y), (x + w, y + h), color, 5)

        label = f"{category_labels.get(category_id, 'Unknown')} ({score})"

        label_background_color = color
        label_text_color = (0, 0, 0) if category_id == 2 or category_id == 4 else (255, 255, 255)

        margin = 10
        (label_width, label_height), baseline = cv2.getTextSize(label, cv2.FONT_HERSHEY_SIMPLEX, 1.0, 2)
        cv2.rectangle(image, (x - margin, y - label_height - 10 - margin), (x + label_width + margin, y + margin), label_background_color, -1)

        cv2.putText(image, label, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 1.0, label_text_color, thickness=3)

    output_path = "output_with_bounding_boxes.jpg"
    cv2.imwrite(output_path, image)
    print(f"Imatge desada a {output_path}")

    image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    plt.figure(figsize=(10, 10))
    plt.imshow(image_rgb)
    plt.axis("off")
    plt.show()

image_path = "/content/EuRoSign/val/images/00445.jpg"
json_path = "/content/drive/MyDrive/TFG/YOLO/resultats/val-C-EuRoSign/predictions.json"
draw_bounding_boxes(image_path, json_path, score_threshold=0.3)

```

```

# Script preparat per mostrar les imatges etiquetades
def draw_yolo_bounding_boxes(image_path, labels_path, class_names, image_size=(640, 640)):

    category_colors = {
        0: (0, 0, 255),    # PROHIBITORY - Blau (BGR)
        1: (0, 255, 255),  # DANGER - Vermell (BGR)
        2: (255, 0, 0),    # MANDATORY - Blau clar (BGR)
        3: (255, 255, 255) # OTHER Blanc (BGR)
    }

    image = cv2.imread(image_path)
    if image is None:
        print(f"No s'ha pogut carregar la imatge de la ruta {image_path}")
        return

    height, width = image.shape[:2]

    try:
        with open(labels_path, 'r') as f:
            labels = f.readlines()
    except FileNotFoundError:
        print(f"No s'ha trobat el fitxer d'etiquetes a la ruta {labels_path}")
        return

    for label in labels:
        label_data = label.strip().split()
        if len(label_data) != 5:
            continue

        class_id = int(label_data[0])
        cx, cy, w, h = map(float, label_data[1:])

        x1 = int((cx - w / 2) * width)
        y1 = int((cy - h / 2) * height)
        x2 = int((cx + w / 2) * width)
        y2 = int((cy + h / 2) * height)

```

```

color = category_colors.get(class_id, (255, 255, 255))
class_label = class_names[class_id] if class_id < len(class_names) else f"Class {class_id}"

cv2.rectangle(image, (x1, y1), (x2, y2), color, 2)

label = f"{class_label}"
margin = 10
label_background_color = color
label_text_color = (0, 0, 0) if class_id == 1 or class_id == 3 else (255, 255, 255)

font_scale = 1.2
text_thickness = 3

(label_width, label_height), baseline = cv2.getTextSize(label, cv2.FONT_HERSHEY_SIMPLEX, font_scale, text_thickness)
cv2.rectangle(image, (x1 - margin, y1 - label_height - 10 - margin),
              (x1 + label_width + margin, y1 + margin), label_background_color, -1)

cv2.putText(image, label, (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX, font_scale, label_text_color, text_thickness)

output_path = "output_with_yolo_boxes.jpg"
cv2.imwrite(output_path, image)
print(f"Imatge desada a {output_path}")

image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
plt.figure(figsize=(10, 10))
plt.imshow(image_rgb)
plt.axis("off")
plt.show()

image_path = "/content/EuRoSign/val/images/00445.jpg"
labels_path = "/content/EuRoSign/val/labels/00445.txt"
class_names = ["PROHIBITORY", "DANGER", "MANDATORY", "OTHER"]

draw_yolo_bounding_boxes(image_path, labels_path, class_names)

```

```

def show_mandatory_images(image_dir, label_dir, class_id=2):
    mandatory_images = []

    image_files = [f for f in os.listdir(image_dir) if f.endswith(('.jpg', '.png'))]

    for image_file in image_files:
        image_path = os.path.join(image_dir, image_file)
        label_path = os.path.join(label_dir, os.path.splitext(image_file)[0] + '.txt')

        if not os.path.exists(label_path):
            continue

        # Comprovem si la detecció pertany a la classe "Mandatory"
        with open(label_path, 'r') as f:
            labels = f.readlines()

        contains_mandatory = any(int(line.split()[0]) == class_id for line in labels)

        if contains_mandatory:
            mandatory_images.append((image_path, label_path))

    num_images = len(mandatory_images)
    num_cols = 3
    num_rows = math.ceil(num_images / num_cols)

    fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, num_rows * 5))
    axes = axes.flatten()

    for i, (image_path, label_path) in enumerate(mandatory_images):
        image = cv2.imread(image_path)
        if image is None:
            print(f"No s'ha pogut carregar la imatge: {image_path}")
            continue

        image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

```

```

image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

with open(label_path, 'r') as f:
    labels = f.readlines()

for label in labels:
    parts = label.strip().split()
    class_id_in_file = int(parts[0])

    if class_id_in_file == class_id:
        x_center, y_center, width, height = map(float, parts[1:])

        img_height, img_width = image.shape[:2]

        # Converteixo les coordenades en píxels
        x_center *= img_width
        y_center *= img_height
        width *= img_width
        height *= img_height

        # Calculo les coordenades de la caixa delimitadora
        x_min = int(x_center - width / 2)
        y_min = int(y_center - height / 2)
        x_max = int(x_center + width / 2)
        y_max = int(y_center + height / 2)

        cv2.rectangle(image_rgb, (x_min, y_min), (x_max, y_max), (255, 0, 0), 3)

    axes[i].imshow(image_rgb)
    axes[i].set_title(os.path.basename(image_path))
    axes[i].axis('off')

for j in range(i + 1, len(axes)):
    axes[j].axis('off')

plt.tight_layout()
plt.show()

image_dir = "/content/EuRoSign/val/images/"
label_dir = "/content/EuRoSign/val/labels/"

show_mandatory_images(image_dir, label_dir, class_id=2)

```

```

# Script per extreure informació dels conjunts

dataset_path = "GTSDB"

image_extensions = {".jpg", ".jpeg", ".png", ".bmp", ".tiff"}

total_images = 0
total_size = 0

for root, dirs, files in os.walk(dataset_path):
    for file in files:
        if os.path.splitext(file)[1].lower() in image_extensions:
            total_images += 1

            total_size += os.path.getsize(os.path.join(root, file))

total_size_mb = total_size / (1024 * 1024)

print(f"Nombre total d'imatges: {total_images}")
print(f"Mida total del dataset: {total_size_mb:.2f} MB")

```