UNIVERSITAT DE BARCELONA

Treball final de grau

GRAU D'ENGINYERIA INFORMÀTICA

Facultat de Matemàtiques i Informàtica
Universitat de Barcelona

Detecció de punts de referència en entorns urbans mitjançant visió per computador

Autor: Aniol Garcia i Serrano

Directors:	Dr. Jordi Vitrià
	Sergi Hernández
	Alejandro López
Realitzat a:	Institut de Robòtica i Informàtica Industrial

Barcelona, 24 de gener de 2023

Resum

L'objectiu d'aquest treball és de proveir al robot autònom ADD d'un sistema de detecció de punts de referència que es pugui integrar fàcilment al mòdul de localització. S'ha desenvolupat diferents mètodes centrant-se en diferents tipus de punts de referència, però aquest treball se centra en l'ús de senyals de trànsit en entorns urbans. Per tal de fer la detecció i classificació dels senyals, es proposen tres mètodes diferents: el primer utilitza una sola xarxa neuronal convolucional (CNN) que fa la detecció i classificació; el segon utilitza una CNN que fa la detecció i una altra que fa la classificació; finalment, el tercer, utilitza una CNN per la detecció i mètodes més tradicionals per fer la classificació. Es consideren diferents xarxes i modificacions per a aquestes tasques, posant èmfasi en les de la família YOLO. Aquests mètodes s'avaluen amb els àmpliament utilitzats datasets GTSDB i GTSRB, on s'obtenen resultats comparables amb els de l'estat de l'art. Posteriorment, s'avaluen sobre entorns molt més similars als que el robot trobarà a partir d'un conjunt de dades creat específicament per aquest fi.

Abstract

This project aims to supply the ADD autonomous robot with a landmark detection system that should be able to easily integrate with the existing localization stack. Multiple landmark have been developed, but this project is centered on urban traffic signs. To be able to detect and classify the signs, three different methods are proposed: the first uses a convolutional neural network (CNN) to both detect and classify the signs; the second one uses a CNN to detect and another one to classify; the third, uses a CNN to detect and more traditional methods to classify. Multiple networks and optimizations are considered, with a strong emphasis with those on the YOLO family. These methods have been tested with the well known GTSDB and GTSRB, and some results have been comparable to the ones obtained in the state of the art. They are also tested in environments much more similar to the ones the robot will encounter, using a specially made dataset.

Resumen

Este proyecto pretende dotar al robot autónomo ADD con un sistema de detección de puntos de referencia que pueda ser fácilmente integrado en el módulo

de localización. Se han desarrollado diferentes métodos para distintos tipos de puntos de referencia, pero este proyecto se centra en el uso de señales de tráfico en entornos urbanos. Para realizar la detección y clasificación de señales se proponen tres métodos distintos: el primero usa una sola red neuronal convolucional (CNN) para la detección y clasificación; el segundo usa una CNN para la detección y otra para la clasificación; finalmente, el tercero usa una CNN para la detección y métodos más tradicionales para la clasificación. Se han considerado múltiples redes y optimizaciones, poniendo énfasis en las de la familia de YOLO. Estos métodos se avaluan usando los conocidos GTSDB y GTSRB, donde los resultados obtenidos son comparables a los del estado del arte. Posteriormente, han sido también testeados en entornos muy similares a los que se encontrará el robot a partir de un conjunto de datos más especializado creado para este fin.

Agraïments

En primer lloc, voldria agrair a l'Institut de Robòtica i Informàtica Industrial, i en especial a Sergi Hernández, Alejandro López i Fernando Herrero per donar-me la possibilitat de realitzar aquest treball, i pel guiatge i l'acompanyament que m'han ofert en tot moment.

També el meu agraïment a en Jordi Vitrià, per la seva confiança a l'hora d'acceptar dirigir-me el treball i, sobretot, pels consells que m'ha anat oferint al llarg del camí.

Finalment, donar les gràcies també a la meva família, per la seva infinita paciència i la seva dedicació.

Gràcies a tots!

Índex

Capítol 0

Introducció

Aquest treball de fi de grau s'emmarca dins el projecte LogiSmile [1], un projecte europeu que estudia la possibilitat de realitzar l'entrega de paqueteria mitjançant vehicles autònoms. L'objectiu del projecte consisteix en realitzar una prova de concepte d'un sistema d'entrega totalment autònom a partir d'un vehicle nodriu autònom (Autonomous Hub Vehicle, anomenat comunament com AHV) que cooperi amb robots més petits (els Autonomous Delivery Device, o ADD), els quals realitzen l'entrega final en entorns urbans. En aquest projecte hi participen múltiples empreses i institucions que s'encarreguen de de la construcció i desenvolupament dels components necessaris. La tasca de programació del robot ADD recau en l'Institut de Robòtica i Informàtica Industrial (IRI), el centre on s'ha desenvolupat aquest treball de fi de grau.

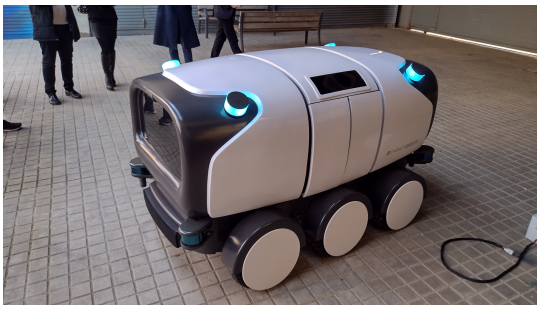


Figura 1: El robot ADD de l'IRI, batejat com a Ona

[1] `https://www.eiturbanmobility.eu/projects/logismile/`

El problema de la localització

Dels components que s'ha de programar, un dels més importants i alhora problemàtics és el mòdul de localització i navegació. Si no es coneix la posició del robot amb un marge d'error suficientment petit, encara que es tingui una bona planificació, no és possible navegar de manera fiable. Aquest problema sembla fàcilment solucionable utilitzant mòduls de GNSS (Global Navigation Satellite System), però en entorns urbans aquest sistema no és prou precís. Aquests sistemes es basen en dispositius que reben unes senyals emeses per satèl·lits que contenen la posició del satèl·lit i el temps exacte en que la senyal va ser enviada. Rebent senyals de com a mínim tres satèl·lits diferents, coneixent la velocitat de transmissió de les ones i el temps de recepció dels senyals, es pot triangular la posició del receptor. Aquest càlcul, però, suposa que el senyal rebut prové directament de l'emissor, cosa que no necessàriament és certa dins la ciutat on pot rebotar entre edificis alts. Aquest problema que es coneix amb el nom de multipath. A més, és possible que hi hagi interferències causades per dispositius que operen en les mateixes freqüències. Tot això provoca que la posició no sigui prou exacte i que sovint faci salts sobtats.

Un altre sistema que s'utilitza és la **odometria**, que consisteix en el càlcul de la posició en funció de la posició inicial i el moviment dels motors fins al moment actual. En teoria, si es coneix com s'han mogut, es poden calcular els desplaçaments i girs, obtenint la posició i orientació final. A la pràctica, no sempre tot el gir del motor es transmet a la roda, les rodes acaben relliscant i les lectures no són prou precises, de manera que sempre s'acaba obtenint una certa deriva.

La manera de solucionar aquests problemes sol ser dotant al robot de múltiples sensors addicionals com LIDAR (Laser Imaging, Detection, and Ranging), càmeres convencionals i de profunditat, unitats inercials, etc. per obtenir característiques de l'entorn i fer-ne la fusió per obtenir millors mesures. Un d'aquests sistemes de fusió és l'SLAM (Simultaneous Localization And Mapping), on el robot va construint un plànol de l'entorn a mesura que l'explora i s'hi situa ell mateix, Concretament, l'ADD utilitza un tipus de SLAM anomenat GraphSLAM, que construeix el plànol utilitzant punts de referència en l'entorn. La idea general és que a partir de múltiples mesures a aquests punts de referència des de diferents posicions estimades, es pot obtenir una millor aproximació de les posicions reals. Per a més informació del funcionament del sistema, es pot utilitzar [24] i [17].

Objectius del treball

Aquest treball tracta precisament la part de trobar punts de referència mitjançant visió per computador. Durant l'estada a l'IRI s'ha investigat múltiples maneres d'obtenir aquests punts de referència, però en aquest treball es tracta principalment la detecció de senyals de trànsit. Els senyals de trànsit són objectes comuns als espais on ha de treballar el robot, són estàtics (i per tant, en el millor dels casos, poden estar geolocalitzats) i estan fets per ser molt visibles. Així doncs, els objectius són els següents:

- Desenvolupar un sistema robust de detecció de senyals de trànsit en zones urbanes.
- Detectar els tipus de senyals més comuns i desenvolupar un mètode que sigui capaç de classificar-los.
- Fer que tots aquests mètodes siguin fàcilment adaptables al robot ADD.
- Entendre la teoria darrere dels mètodes utilitzats.

Estructura de la memòria

La memòria es divideix en dues gran parts: la part teòrica i la part pràctica. A la part teòrica s'introdueixen els conceptes teòrics referents a les eines que s'ha utilitzat, que han estat necessaris per a desenvolupar i entendre tota la part pràctica. El segon capítol és el que parla pròpiament dels senyals de trànsit. A l'inici s'explica l'estat de l'art i els mètodes que s'han anat utilitzant per a resoldre aquest problema i a la segona part els mètodes proposats juntament amb la seva avaluació quantitativa i qualitativa.

Si es fan referències al codi realitzat, sempre seran relatives al directori arrel de codi del projecte, que es pot trobar acompanyant aquesta memòria o bé al repositori `https://gitlab.com/aniolgarcia/vision_landmark`. La major part del codi està escrita en Python i, en general, pel desenvolupament d'algorismes i l'entrenament de xarxes s'utilitzen notebooks de Jupyter, mentre que per la inferència i execució s'utilitzen fitxers `.py`.

Al llarg de la memòria s'ha intentat il·lustrar els conceptes explicats a partir d'imatges i diagrames. Si no s'indica el contrari, totes aquestes imatges són d'autoria pròpia, generades mitjançant els paquets TikZ/PGF o bé llibreries de visualització de Python com Matplotlib.

Capítol 1

Marc teòric

Durant el desenvolupament del treball s'ha utilitzat un gran nombre d'eines i recursos. Amb alguns d'ells ja hi estava familiaritzat, però la majoria no els havia tractat mai o no en coneixia els fonaments teòrics. Un dels objectius del treball ha estat entendre totes aquestes eines, de manera que ha calgut un treball important d'estudi i aprenentatge. En aquest capítol, doncs, s'expliquen alguns dels conceptes teòrics més importants pel desenvolupament del treball.

1.1 Xarxa Neuronal Artificial

L'eina principal que s'utilitza durant el treball són les **xarxes neuronals convolucionals**. Per entendre el funcionament d'aquest tipus de xarxes és convenient fer un pas enrere i estudiar primer xarxes neuronals artificials més senzilles, que són més fàcils de tractar però els principis de funcionament són molt similars. Així doncs, una xarxa neuronal artificial (ANN per les seves sigles en anglès) és un model computacional inspirat en les xarxes neuronals biològiques, capaç de processar informació gràcies a la seva estructura i a diversos algorismes que operen sobre aquesta estructura.

Les unitats bàsiques de les ANN són les neurones que, de manera similar a les neurones biològiques, s'interconnecten entre sí i s'envien senyals entre elles. De manera formal, podem definir una neurona artificial com un conjunt (x, w, b, φ, y), on:

- $x \in \mathbb{R}^n$, amb $x^T = (x_1, \ldots, x_n)$, el conjunt de valors d'entrada
- $w \in \mathbb{R}^n$, amb $w^T = (w_1, \ldots, w_n)$ el conjunt de pesos

- $b \in \mathbb{R}$, anomenat biaix
- $\varphi \colon \mathbb{R} \to \mathbb{R}$ anomenada funció d'activació, normalment contínua i no lineal
- $y \in \mathbb{R}$, amb $y = \varphi(x^T \cdot w) = \varphi\left(b + \sum_{i=1}^{n} x_i w_i\right)$[1]

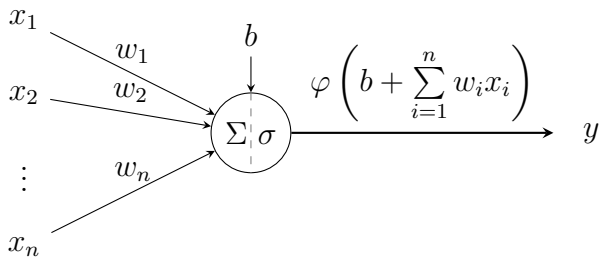


Figura 1.1: Esquema d'una neurona artificial

Aquestes neurones s'organitzen en forma de xarxa, de manera que la sortida d'una neurona s'utilitza com a entrada de les següents. Aquesta estructura acaba formant un graf dirigit amb arestes ponderades on cada node representa una neurona i una aresta dirigida entre dos nodes indica que la sortida de la primera neurona es passa com a entrada a la segona, ponderada pel pes de l'aresta. Les neurones s'organitzen per capes, de manera que el vector format per les sortides y d'una capa s'agafa com a vector d'entrada x de la següent. La primera capa de la xarxa rep els valors externs i s'anomena **capa d'entrada**. L'última capa s'anomena **capa de sortida** i la resta s'anomenen **capes ocultes**.

Definint

- $L \in \mathbb{N}$ el nombre de capes de la xarxa
- $d(l)$ el nombre de nodes de la capa l, amb $1 \leq l \leq L$
- $b^{(l)} \in \mathbb{R}^{d(l)}$ els biaixos de cada capa
- $W^{(l)} \in \mathbb{R}^{d(l) \times d(l+1)}$ la matriu de pesos entre la capa l i la capa $l+1$, amb $W_{ij}^{(l)}$ el pes entre l'i-èssima neurona de la capa l i la j-èssima neurona de la capa $l+1$
- $z^{(l)} \in \mathbb{R}^{d(l)}$ el vector de sumes ponderades de la capa l, $z^{(l)} = W^{(l-1)} a^{(l-1)} + b^{(l)}$

[1]Fent abús de notació, entenent que apliquem φ element a element.

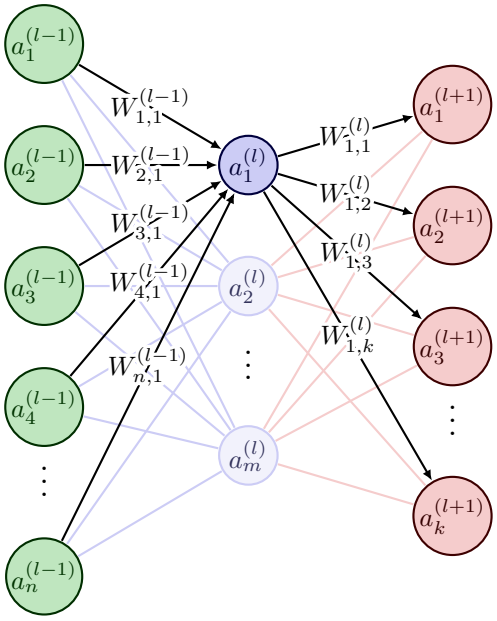


Figura 1.2: Secció de tres capes d'una xarxa neuronal amb èmfasi a la propagació de valors sobre $a_1^{(l)}$.

- $a^{(l)} \in \mathbb{R}^{d(l)}$ la sortida de la capa l (també anomenada activació), amb $a_i^{(l)}$ la sortida de l'i-èssima neurona de la capa i $a^{(l)} = \varphi\left(z^{(l)}\right)$.

podem expressar la propagació de valors entre una capa i la següent com

$$a^{(l)} = \varphi\left(W^{(l-1)}a^{(l-1)} + b^{(l)}\right)$$

que en forma matricial seria:

$$\begin{pmatrix} a_1^{(l)} \\ \vdots \\ a_n^{(l)} \end{pmatrix} = \varphi\left(\begin{pmatrix} W_{1,1}^{(l-1)} & \dots & W_{1,m}^{(l-1)} \\ \vdots & \ddots & \vdots \\ W_{n,1}^{(l-1)} & \dots & W_{n,m}^{(l-1)} \end{pmatrix}\begin{pmatrix} a_1^{(l-1)} \\ \vdots \\ a_m^{(l-1)} \end{pmatrix} + \begin{pmatrix} b_1^{(l)} \\ \vdots \\ b_n^{(l)} \end{pmatrix}\right)$$

amb $n = d(l)$ i $m = d(l-1)$

1.1.1 Funció d'activació

Sense la funció d'activació φ, el vector de valors $a^{(L)}$ obtingut a la capa de sortida seria lineal respecte els valors de la capa d'entrada. Per tant, si es vol que la xarxa representi una funció no lineal, és necessari introduir algun component que no sigui lineal, i aquesta tasca recau en la funció d'activació. A continuació se n'exposen les més importants:

- **Sigmoidal**: Té el problema que se satura molt ràpidament a 0 o 1 i només és sensible a valors d'entrada propers a 0.

$$\begin{aligned} f(x)\colon \mathbb{R} &\longrightarrow (0,1) \\ x &\longmapsto \frac{1}{1+e^{-x}} \end{aligned}$$

- **ReLU**: És de les més utilitzades i té l'avantatge que tant la funció com la seva derivada són molt senzilles de calcular. La derivada és 1 si $x > 0$ i 0 en cas contrari, de manera que és molt senzill propagar-ne el gradient. Té, però, diversos problemes: no és derivable en 0 i no és acotada.

$$\begin{aligned} f(x)\colon \mathbb{R} &\longrightarrow \mathbb{R}^+ \cup \{0\} \\ x &\longmapsto \max(0,x) \end{aligned}$$

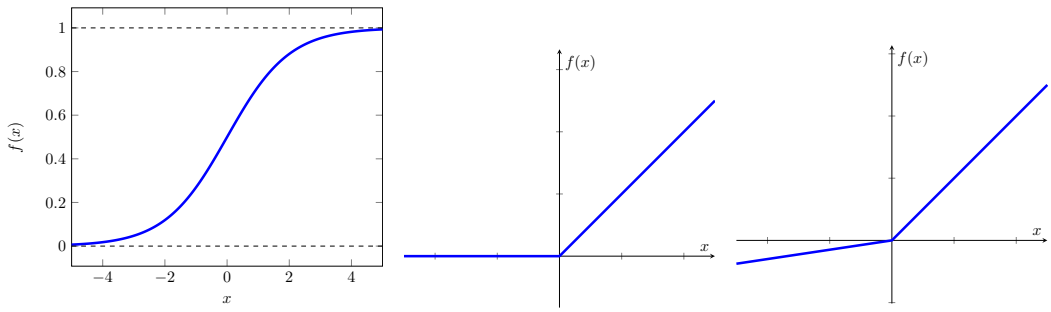
- **Parametrized ReLU**: Aquesta funció va sorgir per solucionar el cas on les sortides d'una capa són sempre negatives, independentment de les entrades. En aquest cas, si es fes servir ReLU, les activacions serien sempre 0, dificultant l'aprenentatge de la xarxa. En aquest cas, en lloc de ser constant en els negatius, s'hi posa una certa pendent, que permetria evitar el problema.

$$\begin{aligned} f_\alpha(x)\colon \mathbb{R} &\longrightarrow \mathbb{R} \\ x &\longmapsto \begin{cases} x & \text{si } x \geq 0 \\ ax & \text{si } x < 0 \end{cases} \end{aligned}$$

  El cas particular amb el paràmetre $\alpha = 0.01$, $f_{0.01(x)}$, s'anomena **Leaky ReLU**.

- **Softmax**: Aquesta funció no s'aplica només a la sortida d'una neurona, sinó que s'aplica sobre les sortides d'una capa sencera, típicament la capa de sortida. El que fa és normalitzar les sortides d'una capa per assimilar-les a probabilitats, de manera que la suma de totes les sortides de la capa sigui 1 i que el valor de cada neurona sigui proporcional a l'exponencial del valor previ a aplicar softmax. Així, si una capa té n neurones i $x = (x_1, \ldots, x_n)^T \in \mathbb{R}^n$ és el vector de sumes ponderades de la capa, es defineix la funció softmax com

$$\begin{aligned} f_i(x)\colon \mathbb{R}^n &\longrightarrow [0,1] \\ x &\longmapsto \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \end{aligned}$$



(a) (b) (c)

Figura 1.3: Gràfics de les funcions Sigmoidal (a), ReLU (b) i Leaky ReLU (c)

1.1.2 Entrenament

Per fer que el comportament de la xarxa s'aproximi al comportament que en volem obtenir, cal ajustar-ne els paràmetres (principalment els pesos i biaixos). Per dur a terme aquest ajust dels pesos normalment es fa servir l'algorisme de **backpropagation**. La premissa de l'algorisme és bastant simple: si d'una entrada en particular en coneixem la sortida que desitgem i la sortida real de la xarxa, podem calcular l'error entre aquestes dues sortides. Prenent l'entrada i sortida constants, podem expressar aquest error com a funció dels pesos i biaixos i, per tant, calcular la derivada parcial de l'error per a cada un dels paràmetres. Aquesta derivada ens donarà una idea de com varia l'error en funció de la variació del paràmetre, de manera que podem veure com hem de canviar cada paràmetre per tal que l'error es minimitzi. El conjunt de les derivades parcials respecte cada paràmetre formen el **gradient** que, pensat com a vector, apunta cap a la direcció de màxim creixement. Aleshores, si el que volem és disminuir l'error, hem de variar els paràmetres en la direcció contrària al gradient, el que anomenem **descens del gradient**.

Formalment, si la xarxa neuronal té L capes i si $C\colon \mathbb{R}^{d(L)} \to \mathbb{R}^{d(L)}$ és una funció derivable tal que $C_y(\hat{y})$ és l'error entre la sortida esperada y i la real $\hat{y}$, el que volem és calcular

$$\frac{\partial C}{\partial W_{ij}^{(l)}} \text{ i } \frac{\partial C}{\partial b_i^{(l)}}$$

Desenvolupant per la regla de la cadena obtenim

$$\frac{\partial C}{\partial W_{ij}^{(L-1)}} = \frac{\partial C}{\partial a_i^{(L)}} \frac{\partial a_i^{(L)}}{\partial z_i^{(L)}} \frac{\partial z_i^{(L)}}{\partial W_{ij}^{(L-1)}}$$

$$\frac{\partial C}{\partial b_i^{(L)}} = \frac{\partial C}{\partial a_i^{(L)}} \frac{\partial a_i^{(L)}}{\partial z_i^{(L)}} \frac{\partial z_i^{(L)}}{\partial b_i^{(L)}}$$

Els dos primers termes dels desenvolupaments són idèntics de manera que, definint $\delta_i^{(L)} = \frac{\partial C}{\partial a_i^{(L)}} \frac{\partial a_i^{(L)}}{\partial z_i^{(L)}}$, les equacions anteriors es poden escriure com

$$\frac{\partial C}{\partial W_{ij}^{(L-1)}} = \delta_i^{(L)} \frac{\partial z_i^{(L)}}{\partial W_{ij}^{(L-1)}}$$
$$\frac{\partial C}{\partial b_i^{(L)}} = \delta_i^{(L)} \frac{\partial z_i^{(L)}}{\partial b_i^{(L)}}$$

El primer terme de $\delta_i^{(L)}$ el podrem calcular directament, ja que si y és la sortida esperada, l'error C serà una funció amb paràmetres y i $\hat{y} := a^{(l)}$. També $a_i^{(L)} = \varphi\left(z_i^{(L)}\right)$, i per tant el segon terme és

$$\frac{\partial a_i^{(L)}}{\partial z_i^{(L)}} = \frac{\partial}{\partial z_i^{(L)}} \varphi\left(z_i^{(L)}\right) = \varphi'\left(z_i^{(L)}\right)$$

En el cas de la derivada de l'error respecte els pesos, per definició $z_i^{(L)} = \sum_{k=1}^{d(L-1)} W_{ik}^{(L-1)} a_k^{(L-1)} + b_i^{(L)}$, de manera que el tercer terme del producte és

$$\frac{\partial z_i^{(L)}}{\partial W_{ij}^{(L-1)}} = \frac{\partial}{\partial W_{ij}^{(L-1)}} \sum_{k=1}^{d(L-1)} W_{ik}^{(L-1)} a_k^{(L-1)} + b_i^{(L)} = a_j^{(L-1)}$$

Procedim de manera similar pel cas $\frac{\partial C}{\partial b_i^{(L)}}$. Els primers dos factors del producte són $\delta_i^{(l)}$ i, per la definició de $z^{(L)}$, tenim que $\frac{\partial z_i^{(L)}}{\partial b_i^{(L)}} = 1$.

Així,

$$\frac{\partial C}{\partial W_{ij}^{(L-1)}} = \delta_i^{(L)} a_j^{(L-1)}$$
$$\frac{\partial C}{\partial b_i^{(L)}} = \delta_i^{(L)}$$

Les expressions de la capa de sortida són prou senzilles, però a mesura que es va retrocedint cap a la capa d'entrada, les expansions de la regla de la cadena

es van complicant, ja que sempre es calcula la derivada parcial de l'error a la última capa i cal anar acumulant totes les composicions. Tot i així, tots els termes que apareixen a l'expansió d'un paràmetre de la capa k excepte l'últim ja s'han calculat en la capa $k+1$, de manera que només cal anar-los guardant i aplicant la següent equació:

$$\delta_i^{(l-1)} = \varphi'\left(z_i^{(l-1)}\right) \sum_{j=1}^{d(l)} W_{ij}^{(l-1)} \delta_j^{(l)}$$

És per aquesta propagació des de la capa de sortida fins a la capa d'entrada dels termes de l'expansió per la regla de la cadena que l'algorisme rep el nom de backpropagation.

Un cop es tenen els gradients calculats, cal modificar-los fent un cert pas cap a la direcció contrària del gradient, és a dir

$$\begin{cases} W_{ij}^{(l)} \leftarrow W_{ij}^{(l)} - \varepsilon \frac{\partial C}{\partial W_{ij}^{(l)}} \\ b_i^{(l)} \leftarrow b_i^{(l)} - \varepsilon \frac{\partial C}{\partial b_i^{(l)}} \end{cases}$$

on ε s'anomena **learning rate**.

Inicialment el descens del gradient es feia per a cada entrada del conjunt de dades pel que s'estava entrenant i s'iterava tot el conjunt múltiples vegades per intentar obtenir convergència. Això, però, és molt costós, i un gradient que pot funcionar bé per a una entrada pot no ser beneficiós per a una altra. A l'altra banda de l'espectre hi ha l'opció de calcular el gradient per a cada una de les dades d'entrenament i fer-ne la mitjana i, només llavors, fer la modificació dels pesos en funció d'aquesta. Per reduir el temps d'entrenament es sol fer servir un terme mig, anomenat **descens del gradient estocàstic**. En aquest procediment, en lloc d'agafar tot el conjunt de dades a cada passada només se n'agafa un subconjunt aleatori i es modifiquen els pesos segons el gradient mitjà dels errors d'aquesta mostra aleatòria, que té molt menys cost computacional. La idea és que aquest gradient mitjà és representatiu de tot el conjunt de dades i, per tant, s'aproxima al gradient mitjà de totes les dades (i com més gran sigui el subconjunt aleatori, millor serà l'aproximació).

Una explicació més rigorosa i formal es pot trobar als dos primers capítols de [5], on també es justifica l'ús de funcions no lineals.

1.2 Xarxes Neuronals Convolucionals

Les xarxes neuronals explicades fins ara es poden fer tan complexes com calgui, afegint tantes capes i neurones com sigui necessari. Això permet fins i tot fer servir imatges com a entrades de la xarxa (sempre que passem l'estructura tridimensional d'amplada, alçada i canals a una dimensió i el nombre de neurones a la capa d'entrada sigui el total de píxels pel nombre de canals), prenent els valors de píxel com a valors d'entrada de la primera capa. Tot i això, està limitada a fer servir capes de les neurones que hem vist fins ara, que no estan adaptades per treballar-hi. Això se soluciona mitjançant les xarxes neuronals convolucionals, que afegeixen diversos tipus de capes útils per treballar amb imatges.

1.2.1 Capa convolucional

Les **convolucions** són unes operacions molt utilitzades en el camp de la visió per computador que consisteixen en, per a cada píxel, fer una suma del valor del píxel i un cert entorn, ponderats per un certs pesos que es descriuen en una matriu anomenada **nucli** (kernel en anglès). A la figura 1.4 se'n mostra un exemple. De manera més formal, si definim

- I la imatge inicial, amb I_{ij} el píxel corresponent a l'i-èssima columna i j-èssima fila.
- K el nucli de dimensions $2n+1$ i $2m+1$. Cal que siguin senars perquè el nucli tingui un centre, que anomenem $K_{0,0}$ i les coordenades s'escriuen referents al centre.
- S la imatge de sortida.

podem escriure la operació de convolució com

$$S_{ij} = \sum_{x=-n}^{n} \sum_{y=-m}^{m} I_{i-x,j-y} \times K_{xy}$$

A través de les convolucions es poden extreure característiques com per exemple contorns o textures. La idea d'afegir capes convolucionals és poder entrenar els pesos del nucli per tal que s'extreguin les característiques més rellevants pel problema que intenta aproximar la xarxa.

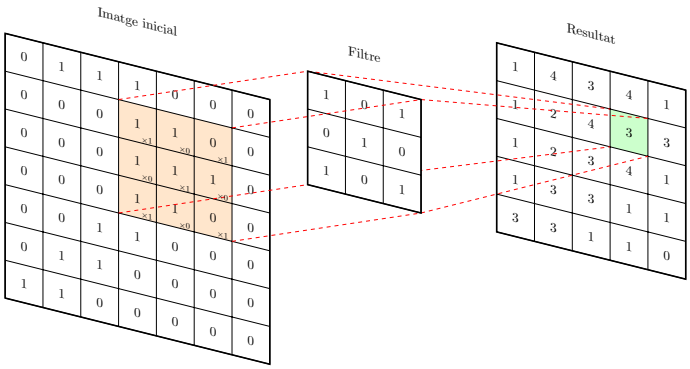


Figura 1.4: Exemple de convolució

1.2.2 Capes de reducció de dimensionalitat o "pooling"

Les imatges d'entrada poden ser de qualsevol mida, però és possible que en diversos punts de la xarxa sigui convenient reduir la mida de les activacions, ja sigui per reduir el nombre de paràmetres o per extreure'n característiques de més alt nivell. En general per fer aquesta reducció es divideix l'entrada en múltiples regions rectangulars sobre les quals es fa la mitjana de tots els valors que conté o bé se'n fa el màxim o mínim. D'aquesta manera, tota la regió queda reduïda a un sol valor.

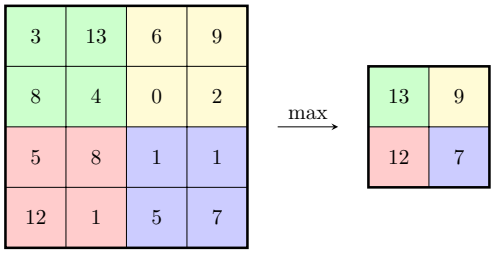


Figura 1.5: Exemple de max pooling

1.2.3 Capes totalment connexes

No són res més que les capes tradicionals de les xarxes neuronals tradicionals. Solen trobar-se cap al final de la xarxa i, arribats a aquest punt, se sol abandonar l'estructura bidimensional de les capes per treballar ja amb vectors similars als que es troben a la capa de sortida.

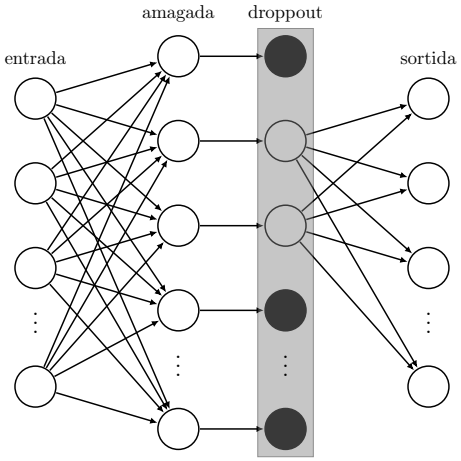


Figura 1.6: Exemple d'una xarxa amb una capa oculta seguida d'una capa droppout amb només dues sortides actives

1.2.4 Capes d'abandonament o "droppout"

Les capes de droppout són un tipus auxiliar de capes que no tenen paràmetres a entrenar, sinó que aporten variabilitat durant l'entrenament. Són capes que fan la funció identitat amb la capa anterior i totalment connexes amb la capa posterior. Tenen la particularitat que, durant l'entrenament, desactiven connexions de manera aleatòria. D'aquesta manera, durant l'entrenament, no sempre es veuen i modifiquen les mateixes capes, de manera que s'intenta evitar l'"overfitting". Se'n pot veure un exemple a la figura 1.6.

1.2.5 Capes de normalització

Definides per primera vegada a [11], les capes de normalització intenten reduir la deriva de la covariància interna de la xarxa. Les sortides de la capa de normalització són simplement les seves entrades ajustades de manera que la mitjana sigui 0 i la seva variància 1, que fa que les sortides siguin molt menys sensibles a grans variacions dels valors d'entrada. La idea d'aquesta capa és que, si la distribució de valors d'entrada d'una capa és prou uniforme durant tot l'entrenament, la distribució de les activacions de la capa també ho serà, i per tant la xarxa convergirà més ràpidament.

1.2.6 Salts

No necessàriament lligats a les xarxes convolucionals, també hi ha xarxes on els valors de sortida d'una capa es poden combinar amb les sortides de capes posteriors. Normalment aquesta combinació és senzillament una suma, però es poden aplicar altres funcions segons la necessitat. La idea darrere dels salts és que si alguna de les capes que es salta perjudica el funcionament de la xarxa, pot disminuir els pesos fins a desactivar-se. A la pràctica, aquests salts també ajuden a minimitzar el problema dels gradients decreixents, ja que si els pensem com capes que fan la funció identitat, el gradient es propaga enrere directament.

1.3 YOLO

YOLO (nom provinent de les sigles de "You Only Look Once") és una xarxa neuronal convolucional que detecta les àrees d'una imatge on hi pot haver certs objectes i els classifica en alguna de les categories desitjades. Tradicionalment, les xarxes que feien detecció i classificació d'objectes utilitzaven dues fases: una on trobaven les àrees d'interès on hi podria haver un objecte, i una altra en què s'analitzaven aquestes àrees. YOLO, però, fa la detecció i classificació en una sola passada, motiu pel qual utilitza aquest nom.

Al llarg d'aquests darrers anys han anat sortint diferents versions i millores, la primera [20], segona [18] i tercera [19] principalment de mans de Joseph Redmond i Ali Farhadi, i a partir de la quarta [4] de manera oficial de mans de Alexey Bochkovskiy, tot i que hi ha hagut altres investigadors que n'han llençat altres versions. En aquest treball s'utilitzen principalment les versions 3 i 4 amb algunes modificacions.

A la xarxa de YOLO, inicialment la imatge d'entrada passa per una sèrie de capes convolucionals que n'extreuen les característiques. Aquesta primera part s'anomena el **"backbone"**, que ha anat evolucionant al llarg de les diferents versions de YOLO. En el cas de YOLO 3, és una xarxa específica anomenada **Darknet53**, i a YOLO 4 **CSPDarknet53**. Un cop han passat per totes les convolucions, els mapes d'activació es divideixen en una quadrícula de dimensió $S \times S$ de caselles idèntiques i cada una de les caselles fa diverses prediccions de **"bounding boxes"**. Com que cada objecte pot estar dividit en múltiples caselles, només la casella on hi ha el centre de l'objecte serà la que farà la predicció de la bounding box de l'objecte. Cada casella produeix B prediccions de bounding boxes i cada predicció és de la forma $(x, y, w, h, p, c_1, \ldots, c_n)$ on:

- $(x, y) \in [0, 1]^2$ són les coordenades del centre de l'objecte, normalitzades a la mida de la casella.
- $(w, h) \in [0, S]^2$ l'amplada i alçada respectivament de la bounding box normalitzades a la mida de la casella. Com que l'objecte pot ocupar més d'una casella, el valor pot anar de 0 fins a S, que és el nombre de caselles que conté la imatge per dimensió.
- $p \in [0, 1]$ la probabilitat de que hi hagi un objecte a la casella, que anomenen **"objectness score"**.
- $(c_1, \ldots, c_n) \in [0, 1]^n$ la probabilitat condicionada que l'objecte detectat sigui de la categoria c_i, suposant que hi ha un objecte a la casella.

A partir de la versió 2 de YOLO s'introdueixen les **"anchor boxes"**, que són uns rectangles d'alçada i amplada predefinides que actuen com a bounding boxes per defecte. Aquesta és una tècnica que ja es feia servir en múltiples xarxes com Faster R-CNN o SSD. Així doncs, a partir de la versió 2, a cada casella se li assignen un nombre d'anchor boxes (5 a la versió 2 i 3 a la versió 3) i per a cada anchor es fa una predicció. Les prediccions segueixen una estructura semblant a les de la versió 1, però amb uns quants canvis per fer els gradients més estables. La xarxa dóna t_x, t_y, t_w, t_h, que són els offsets respecte la bounding box, i es calcula la posició del centre i dimensions reals com

- $b_x = \sigma(t_x) + c_x$
- $b_y = \sigma(t_y) + c_y$
- $b_w = p_w \times e^{t_w}$
- $b_h = p_h \times e^{t_h}$

amb σ la funció sigmoidal definida a l'apartat 1.1.1, (p_w, p_h) les dimensions de l'anchor box (amplada i alçada respectivament) i (c_x, c_y) les coordenades de la casella respecte la quadrícula.

1.3.1 Múltiples escales

Una de les característiques a partir de la tercera versió de YOLO és que es fa la predicció a múltiples escales: no només es divideixen els mapes d'activació en una quadrícula de $S \times S$, si no que es fan tres quadrícules de mides diferents. Específicament, a YOLO 3 i 4 es fa una primera quadrícula amb caselles de

32×32 píxels, una segona divisió amb caselles de 16×16 píxels i un darrera amb caselles de 8×8 píxels (motiu pel qual les imatges d'entrada han de tenir dimensions múltiples de 32). Així, la primera capa és capaç de predir objectes més grans, mentre que cada una de les següents proporciona una granularitat més fina. Com és natural, cada una d'aquestes divisions treballa amb anchor boxes diferents (i normalment també ràtios diferents), per tal d'adaptar-se a les dimensions de les prediccions que s'esperen de cada subdivisió.

1.3.2 Funció de pèrdua

Les arquitectures de YOLO són molt simples en comparació amb altres xarxes convolucionals, però aquesta simplicitat la canvien per una funció de pèrdua molt més complexa que les funcions de pèrdua tradicionals. La funció de pèrdua utilitzada no s'explicita en la publicació [19] sobre la xarxa, però s'ha pogut deduir analitzant el codi publicat[2]. Així, la funció que s'utilitza és la següent:

$$
\begin{aligned}
&\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} (2 - w_{ij} h_{ij}) \left[(x_{ij} - \hat{x}_{ij})^2 + (y_{ij} - \hat{y}_{ij})^2 + (w_{ij} - \hat{w_{ij}})^2 + (h_{ij} - \hat{h_{ij}})^2 \right] \\
&+ \lambda_{obj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} \left[\hat{C_{ij}} log(C_{ij}) + (1 - \hat{C_{ij}}) log(1 - C_{ij}) \right] \\
&+ \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{noobj} \left[\hat{C_{ij}} log(C_{ij}) + (1 - \hat{C_{ij}}) log(1 - C_{ij}) \right] \\
&+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{obj} \sum_{c \in classes} \hat{p_i}(c) log\left(p_i(c)\right) + \left(1 - \hat{p_i}(c)\right) log\left(1 - p_i(c)\right)
\end{aligned}
$$

on

- $\mathbb{1}_{ij}^{obj}$ indica la presència d'un objecte a la cel·la i i que l'anchor j és el responsable de la seva detecció, mentre que $\mathbb{1}_{ij}^{noobj}$ no hi ha cap objecte que l'anchor j de la cel·la i hagi de detectar.
- $x_{ij}, y_{ij}, w_{ij}, h_{ij}$ indiquen els offsets reals respecte la j-èssima anchor box de la i-èssima casella i $\hat{x}_{ij}, \hat{y}_{ij}, \hat{w}_{ij}, \hat{h}_{ij}$ n'indiquen les prediccions.

[2] `https://github.com/AlexeyAB/darknet/`

- $\hat{C}_{ij}$ la probabilitat de si hi ha o no un objecte a la j-èssima anchor box de la i-èssima casella (objectness score).
- C_{ij} és 1 si realment hi ha algun objecte i 0 en cas contrari.
- $\hat{p}_i(c)$ probabilitat de que l'objecte detectat a la casella i sigui de la classe c.

Aleshores, el primer terme és la suma dels quadrats de l'error dels offsets de l'anchor. El segon i tercer terme fan referència a l'existència d'objectes i utilitzen Binary Cross Entropy. El segon terme penalitza l'error de les cel·les que realment contenen un objecte i el tercer el de les cel·les que no n'han de contenir cap. El quart terme només es calcula en les cel·les on s'ha de trobar algun objecte i penalitza l'error de la tria de classe, també amb Binary Cross Entropy.

1.4 Mètriques d'avaluació

Quan es parla de xarxes neuronals, per donar una idea sobre com de bé funcionen sobre un problema concret, és necessari establir una sèrie de mètriques. Aquestes mètriques avaluen el funcionament de la xarxa sobre una sèrie d'entrades on es coneix prèviament la sortida esperada. La mètrica més utilitzada en les xarxes de reconeixement d'objectes és la **"Mean Average Precision"** (normalment estilitzat com **mAP**). Per entendre el concepte de mAP calen els següents conceptes previs:

- Positius Veritables (TP per les sigles en anglès): Es dóna quan la xarxa fa una detecció corresponent a la sortida esperada.
- Falsos Positius (FP): quan la xarxa fa una detecció quan realment no hi ha cap objecte.
- Negatius Veritables (TN): quan la xarxa no fa cap detecció i realment no hi ha cap objecte a detectar (en detecció d'objectes normalment no es fan servir, ja que la major part de les possibles sortides són Negatius Veritables).
- Falsos Negatius (FN): Quan la xarxa no detecta un objecte on realment sí que hi és.

Ara bé, cal definir exactament què significa que la detecció correspongui o no a la sortida esperada, ja que en detecció d'objectes es prediu una capsa contenidora amb un cert centre i dimensions. Sovint, però, hi ha múltiples capses que contenen completament l'objecte i que es podrien donar com a vàlides. Primer cal recordar que les sortides de les xarxes neuronals solen donar una "probabilitat"de la detecció i, per tant, s'ha de definir un valor límit a partir del qual es considera la predicció com a bona. Un cop se sap quines prediccions descartar i quines agafar com a bones, per avaluar la predicció s'utilitza l'"intersection over union"(estilitzat com a **IoU**), on es calcula l'àrea de la intersecció i la unió de la capsa predita i l'esperada i se'n fa el quocient. Si aquest quocient està per sobre d'un cert valor preestablert, es considera que la detecció és correcta.

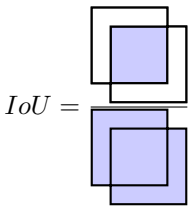


Figura 1.7: Representació del càlcul d'IoU

Aleshores, es defineixen els següents termes:

- **Exactitud** = $\frac{TP+TN}{TP+TN+FP+FN}$
- **Precisió** = $\frac{TP}{TP+FP}$
- **Exhaustivitat** = $\frac{TP}{TP+FN}$

A vegades també es dóna l'exactitud com a mètrica per sí sola, i a partir de la mitjana harmònica de la precisió i exhaustivitat es defineix la mètrica **F1**:

$$F1 = 2 \cdot \frac{\text{precisió} \cdot \text{exhaustivitat}}{\text{precisió} + \text{exhaustivitat}}$$

La precisió dóna una idea sobre l'exactitud del model quan fa una detecció. Una precisió alta es dóna quan el nombre de positius veritables és significativament major que el nombre de falsos positius o bé quan hi ha molt pocs falsos positius.

L'exhaustivitat dóna una idea de com d'exhaustiu és el model alhora de fer deteccions. L'exhaustivitat serà alta quan el model faci la majoria de

deteccions que s'esperen, encara que realment estigui fent moltes deteccions falses d'objectes.

Aleshores, per a cada classe es calcula TP, FP, TN i FN sobre tot el conjunt de dades d'avaluació, però considerant, a més, les deteccions a partir de múltiples valors límit de probabilitat. Per exemple, per a la primera classe es calcula TP, FP, TN i FN prenent com a deteccions bones totes aquelles amb una probabilitat > 0.1, després es recalculen els valors prenent les deteccions > 0.2, etc. fins a arribar a 1. Si aquest procediment es fes per a infinits valors de probabilitat en el rang $[0, 1]$, s'obtindria la corba Precisió-Exhaustivitat per a la primera classe. Aleshores, si anomenem a la corba Precisió-Exhaustivitat de la classe i $P_i(\tau)$, es defineix l'"average preccision"(**AP**) de la classe i com

$$AP_i = \int_0^1 P_i(\tau) d\tau$$

Ara bé, a la pràctica no es pot avaluar infinits punts de probabilitat, de manera que s'agafen n punts equidistants entre 0 i 1 (normalment $n = 11$). Si $\Delta\tau = \frac{1}{n}$, aleshores

$$AP_i = \sum_{k=1}^{n} \Delta\tau \cdot P_i(k)$$

Finalment, quan això s'ha fet per les c classes possibles, es pot calcular l'mAP com

$$mAP = \frac{1}{c} \sum_{i=1}^{c} AP_i$$

Durant tot aquest procés, el valor límit d'IoU s'ha mantingut constant, però també es pot calcular per a diferents valors d'IoU. Per aquest motiu normalment s'especifica quin valor s'ha fet servir. Per exemple, si s'ha utilitzat $IoU > 0.5$, s'escriuria **mAP**0.5.

1.5 Màquines de vectors de suport

A baix nivell, el que fan moltes xarxes neuronals és realment aprendre les característiques rellevants per les dades d'entrada i classificar segons aquestes. Aquest procés tradicionalment es feia manualment, seleccionant les característiques, extraient-ne un descriptor i classificant segons el descriptor. Un dels mètodes més utilitzats per fer la classificació dels descriptors són les màquines de vectors de suport, i és el sistema que s'utilitza en un dels mètodes proposats.

Les **màquines de vectors de suport** (escrit com SVM, de l'anglès Support Vector Machines) són models d'aprenentatge supervisat que divideixen l'espai de paràmetres en dues regions corresponents a dues categories diferents. Per fer-ho, es busca l'hiperplà tal que es maximitza la distància al punt més proper de les dues classes.

Formalment, partim d'un conjunt de dades etiquetades $(x_1, y_1), \ldots, (x_k, y_k)$, amb $x_i \in \mathbb{R}^n$ les mostres d'entrenament i $y_i \in \{1, -1\}$ la classe de la mostra i volem trobar un hiperplà tal que la distància mínima als x_i de cada classe sigui màxima. Tot hiperplà d'un espai n-dimensional es pot escriure com $w \cdot x - b = 0$, amb $w \in \mathbb{R}$ el vector normal al pla. Suposant que es pogués separar completament les dues categories (cosa que no sempre és possible) volem trobar valors w i b tals que els dos hiperplans paral·lels H_1 i H_2 amb $H_1 : w \cdot x - b = 1$ i $H_2 : w \cdot x - b = -1$ separin les dades i la distància entre ells sigui la màxima possible. Suposant que H_1 i H_2 separessin les dades, es compliria que $x \cdot x_i - b > 1$ si $y_i = 1$ i que $w \cdot x_i - b < 1$ si $y_i = -1$. Aleshores, també seria cert que $y_i(w \cdot x_i + b) - 1 \geq 0,\ \forall i$.

La distància entre H_1 i H_2 ve donada per $\frac{2}{||w||_2}$, de manera que, per maximitzar-la, es tracta de minimitzar $||w||_2$. En ser la norma euclidiana, $||w||_2$ conté una arrel quadrada, així que a la pràctica se sol minimitzar el seu quadrat. El problema de minimització queda

$$\min_{w,b} ||w||_2^2 \quad \text{tal que} \quad y_i(w \cdot x_i + b) - 1 \geq 0,\ \forall i$$

que se sol resoldre a partir de multiplicadors de Lagrange.

En cas de que el conjunt de dades no sigui separable, es poden introduir uns certs modificadors $\zeta_i > 0$, $i = 1, \ldots, k$ i adaptar-ne les restriccions punt a punt de la següent manera

$$y_i(w \cdot x_i + b) - 1 \geq \zeta_i,\ \forall i$$

Per compensar aquests canvis, la funció a minimitzar passaria a ser

$$||w||_2^2 + C \sum_{i=0}^{k} \zeta_i$$

per una certa C. En ambdós casos, per classificar una nova mostra només cal mirar a quin costat del pla $w \cdot x - b$ cau, és a dir

$$f(x) = sign(w \cdot x - b)$$

Aquest mètode només serveix per separar les dues classes, però podem reduir el problema de classificació de múltiples classes en múltiples problemes de classificació binària. Hi ha dues maneres essencials de fer-ho:

- **Una contra totes**: Es construeix un classificador per a cada classe on se separa la classe específica de tota la resta de mostres. Per fer inferència, es passa la mostra a tots els classificadors i es determina la classe pel detector que hagi obtingut una millor resposta.
- **Totes contra totes**: Es fan classificadors que separen les classes dues a dues. Per fer-ne la inferència, es torna a passar la mostra a cada classificador i s'assigna la classe que hagi obtingut més nombre de respostes positives.

Capítol 2

Detecció i classificació de senyals de trànsit

2.1 Estat de l'art

En general, la detecció de senyals presenta múltiples dificultats: la majoria de les imatges amb què es treballa són preses des d'un vehicle en moviment i sota condicions d'il·luminació molt variables; els senyals solen ser petits respecte la mida de la imatge, i les imatges poden ser bastant complexes. A més, hi ha afegida la dificultat que tot sistema que es vulgui implementar en vehicles en moviment ha de funcionar pràcticament en temps real, que ja descarta múltiples possibilitats.

La part positiva és que, en ser infraestructura crítica i necessària per la circulació, els senyals normalment són de formes geomètriques predefinides i de colors vius i fàcils de distingir, que facilita el fet de localitzar-los dins la imatge i determinar quin senyal és. Aquestes són, de fet, les dues fases principals dels algorismes de reconeixement de senyals:

- La fase de **localització**, que consisteix en determinar una àrea d'interès dins la imatge on hi ha un senyal.
- La fase de **classificació**, que consisteix en determinar quina senyal concretament hi ha dins aquesta regió.

Hi ha investigadors [16] que hi afegeixen una tercera fase consistent en determinar la tipologia del senyal detectat, sovint diferenciant entre senyals d'obligació, prohibició i perill. Aquesta tercera fase, però, sovint ja s'engloba dins la fase de classificació o bé es classifiquen directament en les categories finals.

Per la fase de detecció normalment es fa la segmentació de la imatge per color, utilitzant el coneixement que es té a priori sobre els colors dels senyals. Per fer-ho, a [12] utilitzen una relació entre les intensitats de cada canal RGB i la suma de tots els canals, i a [7] calculen una altra relació en funció de la intensitat d'un dels canals RGB. En general, però, el sistema de representació de colors RGB no és massa utilitzat en aquest camp, ja que els valors l'espectre de color està distribuït entre els tres canals i és molt sensible a canvis d'il·luminació, cosa que fa que sigui molt difícil establir límits fixes pels valors d'un color, i s'hagi de fer relacions més complexes com les que es detallen en aquests dos articles. També a [13] han utilitzat els sistemes de representació YUV i a [21] el sistema LAB definint valors límit per a cada color rellevant, però sens dubte el sistema més utilitzat és l'HSV. Per evitar establir límits fixes (**"hard thresholds"**) als valors de canals HSV, [6] i [8] fan servir **"Lookup Tables"** (LUTs) per fer una transformació prèvia sobre els canals i establir els límits sobre els valors transformats. Yang et al. [25] calculen, per a cada píxel, la probabilitat de pertànyer al color objectiu, obtenint una imatge de probabilitats que després es binaritza.

En tots aquests sistemes, s'acaba obtenint una imatge binària de les mateixes dimensions que la original amb 1 a les zones amb colors dins el rang establert i 0 a la resta, obtenint una imatge per a cada color a detectar. És necessari, doncs, extreure'n les zones d'interès de la imatge on hi pot haver senyals. Per fer-ho, inicialment s'utilitzaven tècniques de pattern matching fent cross-correlation, però en no ser invariants respecte l'escala, el més comú és utilitzar components connexes i trobar les bounding boxes que s'utilitzaran com a regions d'interès.

També hi ha treballs que descarten la informació de color i treballen a partir de detectors de contorn o formes (com la transformada de Houg per a cercles) i, fins i tot, algorismes basats en la detecció de simetries radials com [2]. L'avantatge d'aquests algorismes és que solen ser molt més robusts a variacions d'il·luminació, però sovint són més costosos que els basats en color.

Un cop detectades les regions on hi pot haver senyals, es fa la classificació, on també s'ha utilitzat molts mètodes diferents. Per exemple a [16] es desenvolupen una sèrie d'algorismes que determinen la forma del senyal buscant cantonades a posicions concretes de les regions trobades i, juntament amb la informació de color de l'apartat de detecció, en fan una classificació preliminar en les categories de perill, cedir el pas, informació, obligació, prohibició i stop. A continuació n'extreuen el contorn del pictograma i en calculen un descriptor, que es compara amb una base de dades precalculada. A [3] s'utilitza una support vector machine (SVM) a la que se li passa la regió detectada en escala

de grisos. Hi ha, també altres treballs que no passen la imatge directament a l'SVM, sinó que abans n'extreuen descriptors mitjançant HOG (histogrames de gradients orientats) o característiques de Harr. Cada vegada més, però, es va veient un ús més extensiu de les xarxes neuronals, com per exemple a [6], que utilitzen una xarxa neuronal per fer la classificació.

De fet, el camp de la visió per computador s'ha vist molt beneficiat per l'ús de xarxes neuronals, que permeten fer l'extracció de característiques de manera automàtica sense dependre d'algorismes amb valors ajustats manualment. Així, la major part de la investigació en aquest camp s'ha centrat en l'ús de xarxes neuronals, tant per la fase de detecció com la de classificació. Tavernik i Skocaj [23] proposen un sistema que fa detecció i classificació utilitzant Mask R-CNN, i a [15] apliquen l'evolució de Mask R-CNN, Faster R-CNN. A [1] es fa l'avaluació de múltiples xarxes com YOLOv2, Faster R-CNN amb diversos backbones, SSD també amb diferents backbones i R-FCN. Finalment a [9], una de les publicacions més recents que tracta aquest tema s'utilitza una combinació de YOLOv4 amb Mobilenet que anomenen YOLO-ML.

2.2 Treball realitzat

Per tal de realitzar la detecció dels senyals de trànsit es proposen tres mètodes diferents:

- **Mètode 1: Una sola xarxa neuronal**. En aquest mètode es fa servir una sola xarxa neuronal on es passa la imatge obtinguda i fa tant la detecció com la classificació dels senyals.
- **Mètode 2: Dues xarxes neuronals**. En lloc de fer servir una única xarxa neuronal, se'n fa servir dues: una primera que fa la detecció de les regions d'interès on hi pot haver senyals i la segona que rep únicament aquestes regions i fa la classificació dels senyals que conté.
- **Mètode 3: Una xarxa neuronal i tècniques tradicionals**. De manera similar a l'anterior mètode, hi ha una xarxa que troba les regions d'interès, però la classificació es fa mitjançant tècniques més tradicionals d'anàlisi de color, extracció de característiques i classificació.

2.2.1 Conjunts de dades utilitzats

Tal com ja s'ha vist a la secció 1.1.2, és el conjunt de dades que s'utilitza durant l'entrenament el que condiciona com s'ajusten els pesos de la xarxa, de manera

que és molt important que el conjunt de dades sigui adequat a l'ús que es vol donar a la xarxa. A continuació s'explica els conjunts de dades utilitzats.

GTSDB

Publicat per l'"Institut Für Neuroinformatik"de la universitat de Ruhr, el El "German Traffic Sign Detection Benchmark"(conegut per les sigles GTSDB)[10] conté un total de 900 imatges, ja separades en un conjunt d'entrenament de 600 imatges i un conjunt d'avaluació de 300. Les imatges són de vistes frontals preses des d'un cotxe en diferents zones d'Alemanya, tant urbanes com carreteres interurbanes i autopistes, i en diferents hores del dia (i per tant diferents condicions d'il·luminació). Cada imatge inclou fins a 6 senyals amb anotacions dels rectangles que els contenen, determinats per les coordenades de la cantonada superior esquerra i la cantonada inferior dreta. A part de la posició dins la imatge també es dóna la categoria del senyal que conté, d'un total de 43 categories. A la figura 2.1 es mostra la quantitat d'imatges que hi ha per a cada categoria.

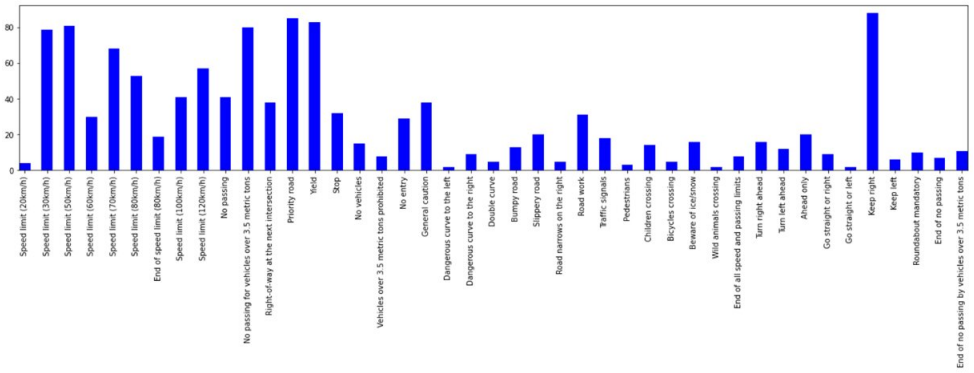


Figura 2.1: Gràfic de distribució de categories de GTSDB. A l'eix horitzontal hi ha les categories i a l'eix vertical el nombre d'imatges per categoria

Aquest conjunt de dades és el que s'utilitza en la gran majoria d'articles que tracten el problema de la detecció de senyals, de manera que es va prendre com a referència per entrenar els detectors. Malgrat tot, aquesta tria va comportar certs problemes i dificultats.

Mirant per sobre les imatges, ja es pot veure que una part significativa són de carreteres interurbanes i autopistes, mentre que tots els tests amb el robot s'han realitzat per zones urbanes. Aquesta diferència d'entorn ja condiciona molt les senyals que poden aparèixer: el nostre sistema no ha trobat mai una senyal de limitació de velocitat de més de 30 km/h mentre que GTSDB no conté

la senyal de limitació a 10 km/h però sí totes les de 50 km/h fins a 120 km/h; La senyal de pas de vianants és de les que més freqüentment hi ha a les zones urbanes, però GTSDB no les anota (i, per tant, durant la fase d'entrenament del detector les deteccions d'aquest senyal es contaven com a falsos positius!); En zones urbanes rarament apareixeran senyals que alerten del pas de bestiar o bé de perill de carretera glaçada, o ni tan sols la de prohibició d'avançar; etc.

Una altra observació potser més subtil i sorprenent és que els senyals d'Alemanya no necessàriament són representatius dels que trobem aquí. Més enllà de les variacions dels pictogrames que s'utilitzen a l'interior dels senyals, hi ha també diferències més conceptuals. Per exemple, basant-nos només amb les imatges de GTSDB s'ha observat que a Alemanya sovint no prohibeixen girar cap a la dreta, sinó que obliguen a continuar recte o girar cap a l'esquerra. A l'àrea metropolitana de Barcelona, però, la tendència és la contrària, de manera que s'hi troben molts senyals de prohibició que a GTSDB estan infrarepresentats senzillament perquè no apareixen a les carreteres alemanyes.

GTSRB

També publicat per "Institut Für Neuroinformatik", hi ha el "German Traffic Sign Recognition Benchmark"[22], conegut com a GTSRB. Aquest conjunt de dades conté cap a 50.000 imatges amb un únic senyal que n'ocupa la major part, classificades en les mateixes 43 categories que s'utilitzen a GTSDB. Al utilitzar aquesta mateixa classificació, aquest conjunt de dades hereta la mateixa problemàtica del GTSDB, i hi trobem una distribució de senyals que no necessàriament és representativa de l'entorn on s'ha de treballar.

Tal com l'anterior, aquest conjunt de dades és àmpliament utilitzat quan es tracta la classificació de senyals de trànsit, de manera que també es va prendre com a referència.

Vídeos de l'ADD

Malgrat no ser un dataset com a tal, les pròpies imatges obtingudes pel robot ADD en tests sobre el terreny també s'han utilitzat molt. Són vídeos de circulació tant per carretera com per zones de vianants en entorns on el robot haurà de treballar i amb la mateixa càmera que s'utilitzarà. Malgrat no haver-hi anotacions dels senyals, aquests són els vídeos sobre els que s'han executat tant els mètodes de detecció com de classificació per fer-ne l'avaluació.

Nou conjunt de dades

Degut a la disparitat de senyals utilitzats als conjunts de dades alemanys i els que hi ha en els entorns on ha de funcionar el robot, es va generar un nou conjunt de dades que fos més representatiu. Els classificadors entrenats amb GTSRB els faltaven múltiples categories i, en trobar algun senyal que no havien vist durant l'entrenament, es classificava erròniament. Així, és necessari que durant l'entrenament es vegin elements de cada classe, i per tant s'ha d'ampliar la base de dades amb totes aquestes noves classes. Un cop es va obtenir un detector de senyals que funcionava prou bé, es va executar el detector sobre alguns vídeos de l'ADD (reservant-ne d'altres per a l'avaluació) per extreure'n les capses contenidores trobades. Posteriorment es va analitzar les imatges, identificant els senyals més comuns i representatius i es van classificar manualment, tot generant un nou conjunt de dades. A la taula 2.1 s'especifiquen les categories utilitzades.

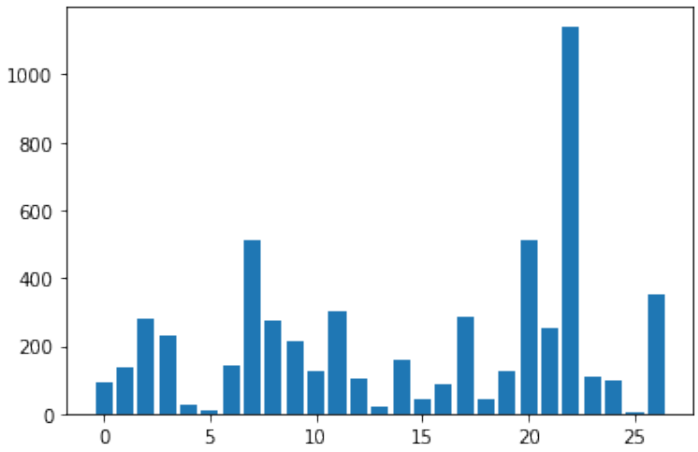


Figura 2.2: Nombre d'imatges per classe

Com que aquest conjunt de dades es va generar cap a finals del projecte, ja amb una visió de com es comportaven els classificadors en entorns reals, a aquestes categories es va afegir la categoria `no_senyal`, que conté exemples de falsos positius trobats pel detector.

A continuació s'expliquen els tres mètodes utilitzats per fer la detecció i classificació de senyals a partir de tots aquests conjunts de dades.

Id	Senyal	Color
0	Limitació tonatge	Vermell
1	Aparcament	Blau
2	Calçada amb prioritat	Groc
3	Cediu el pas	Vermell
4	Ciclistes	Vermell
5	Fi de calçada amb prioritat	Groc
6	Límit 10 km/h	Vermell
7	Límit 30 km/h	Vermell
8	No senyal (falsos positius)	Blau, Groc, Vermell
9	Obligació esquerra/dreta	Blau
10	Obligació esquerra/dreta endavant	Blau
11	Obligació esquerra o dreta	Blau
12	Obligació recte	Blau
13	Perill obres	Vermell
14	Pas escolar	Vermell
15	Pas sense prioritat	Vermell
16	Pas de vianants	Blau
17	Prohibit aparcar	Blau, Vermell
18	Prohibit camions	Vermell
19	Prohibit circular	Vermell
20	Prohibit girar a esquerra/dreta	Vermell
21	Prohibit parar i estacionar	Blau, Vermell
22	Prohibit passar	Vermell
23	Precaució pas elevat	Vermell
24	Rotonda	Blau
25	Perill semàfor	Vermell
26	Stop	Vermell

Taula 2.1: Classes i colors del nou conjunt de dades

2.2.2 Mètode 1: Una sola xarxa neuronal

La primera idea va ser agafar una xarxa neuronal especialitzada en detecció d'objectes i entrenar-la amb el dataset GTSDB per fer la detecció i classificació de senyals simultàniament, tal com es fa en els últims desenvolupaments en el camp.

El primer pas va ser muntar els entorns de desenvolupament necessaris per entrenar i fer inferència sobre diverses xarxes i intentar replicar els resultats que ja s'havien obtingut amb sistemes similars. Partint sobretot dels resultats obtinguts a [1] i [9], que són alguns dels articles que tracten els desenvolupaments més recents, es va optar per fer l'entrenament de les següents xarxes: YOLOv3, YOLOv4, SSD i Faster R-CNN.

Tant YOLO v3 com YOLO v4 tenen la implementació "oficial"sobre el fra-

mework Darknet [1], implementat sobre C i CUDA. Aquestes xarxes són especialment interessants, ja que es pot fer inferència sobre els models ja entrenats carregant-los al mòdul DNN[2] d'OpenCV. Com que el robot funciona amb el sistema ROS i un maquinari específic, no és trivial instal·lar programari i llibreries arbitràries, ja que sovint hi ha conflictes de versions entre el programari que es vol instal·lar i els mòduls necessaris pel funcionament del robot que ja hi són. OpenCV, però, és un component que ja hi està integrat i el sistema té les llibreries necessàries per utilitzar-lo en GPU, de manera que no caldria introduir cap component nou.

SSD i Faster R-CNN, per altra banda, s'han implementat sobre el framework PyTorch[3], on també ha estat necessari programar les funcions d'entrenament i inferència.

Cada framework té les seves convencions pel que fa a els formats d'entrada de les imatges i anotacions, de manera que ha calgut adaptar les dades de GTSDB als requeriments de cada un.

En el cas de les xarxes YOLO, els requeriments són els següents:

1. **Mida**: Tal com s'ha explicat a l'apartat 1.3.1, per tal de poder fer la detecció a múltiples escales és necessari que les imatges tinguin mides d'amplada i alçada múltiples de 32, però les imatges de GTSDB tenen mida 1360×800. La resolució amb la ràtio més semblant que compleix la restricció de la multiplicitat és la de 1280×736, de manera que es va fer la conversió de les imatges a aquesta mida i es va adaptar la capa d'entrada de les xarxes per acceptar imatges d'aquesta mida.

2. **Anotacions**: Yolo espera les coordenades del centre i dimensions de la capsa contenidora, relatives a la mida de la imatge, però les anotacions de GTSDB només donen les coordenades absolutes de la cantonades superior esquerra i inferior dreta del senyal.

3. **Estructura del directori**: Per entrenar una xarxa amb el framework de Darknet, a part de les imatges, es requereixen els següents fitxers:

    - Els fitxers `train.txt` i `test.txt` amb els camins a totes les imatges que es vol fer servir per entrenar i avaluar la xarxa respectivament.
    - Un fitxer `*.names` que conté l'etiqueta/nom d'una classe a cada línia.

[1] `https://pjreddie.com/darknet/`
[2] `https://docs.opencv.org/4.x/d6/d0f/group__dnn.html`
[3] `https://pytorch.org/`

- Un fitxer `*.data` amb el nombre de classes que s'utilitzarà i els camins als tres fitxers anteriors.

A part d'aquests fitxers, per a cada imatge del conjunt d'entrenament o avaluació s'ha de crear un fitxer de text amb el mateix nom i mateix directori que contingui les anotacions per aquella imatge.

PyTorch té una solució de més alt nivell que no requereix modificar el dataset ni la seva estructura de directoris, i fa implementar en python una classe que heredi de la classe abstracta `torch.utils.data.Dataset`[4]. Aquesta classe és l'encarregada de llegir les imatges i anotacions del conjunt de dades i fer les transformacions necessàries abans de retornar-les.

Un cop el dataset va ser adaptat es van haver programat totes les funcions auxiliars necessàries, es va procedir a l'entrenament dels models amb les 43 classes de GTSDB. Per l'entrenament de totes les xarxes es fa fer 2000 epochs, utilitzant stochastic gradient descent.

Els notebooks de Jupyter utilitzats per l'entrenament es troben a `traffic_signs/full_CNN/{yolo/ssd/faster-rcnn}_training.ipynb`.

Hi va haver certs problemes amb l'entrenament de YOLO v4 on, a la iteració 1000, es llençava una excepció de la llibreria CUDDN, necessària per entrenar les xarxes en GPU, i després d'intentar diagnosticar i solucionar el problema, es va desistir. Posteriorment es va veure que el problema era degut a un error conegut[5] referent a certes versions de CUDDN que encara no s'ha sol·lucionat.

Els resultats de l'avaluació dels models es mostren a la taula 2.2.

Model	**mAP0.5**
YOLO v3	0.41
YOLO v4	-
Faster R-CNN	0.609
SSD	0.32

Taula 2.2: Resultats de l'entrenament amb 43 classes

[4]Veure `https://pytorch.org/docs/stable/data.html#torch.utils.data.Dataset`

[5]Veure `https://github.com/AlexeyAB/darknet/issues/8669` i `https://github.com/AlexeyAB/darknet/issues/8667`

Experiments sobre dades reals

Com es pot veure, els resultats no van ser massa esperançadors: un model que no es pot entrenar i dos que no arriben al 50% de **mAP0.5**. Faster R-CNN funciona molt millor que les altres, però a la pràctica no és suficient per fer una classificació fiable. En executar aquestes xarxes sobre les imatges de l'ADD, aquest mal resultat s'accentua. Troba bastants senyals, però també se'n deixa molts, hi ha molts falsos positius i moltes classificacions errònies. El fet és que segurament aconseguir un bon resultat amb aquest mètode era una mica ambiciós: detectar i classificar 43 categories diferents, d'objectes petits i a sobre molt similars entre ells, amb només una mostra de 600 imatges amb una distribució molt poc uniforme... De fet, ni a [1] ni [9] fan la classificació en les 43 categories diferents, sinó que es limiten en classificar-les en 4 supercategories (obligació, prohibició, perill i altres). És a dir, que s'aturen a la fase auxiliar de la que es parlava a 2.1. Aquest va ser, doncs, el següent objectiu.

2.2.3 Mètode 2: Dues xarxes neuronals

Veient que al mètode 1 la detecció era la part que millor funcionava, es va decidir separar el procediment en dues parts: la detecció i la classificació.

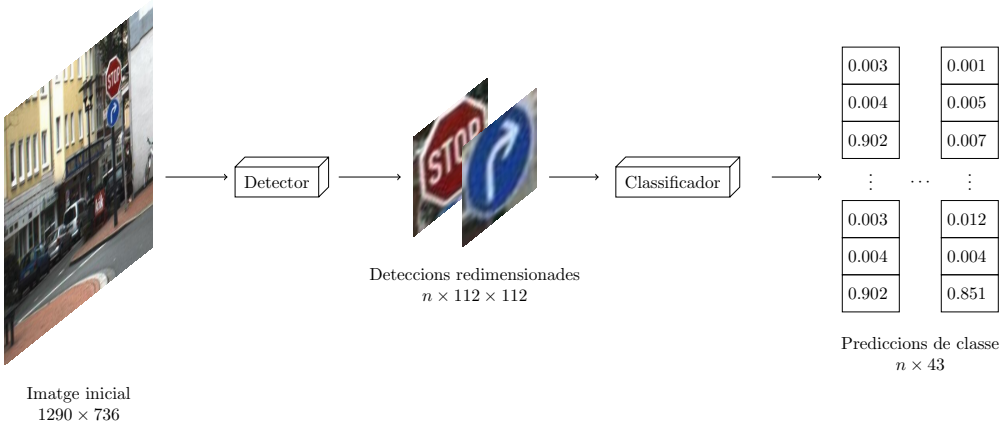


Figura 2.3: Diagrama de funcionament del mètode

Supercategoria	Classes
Prohibició (0)	0, 1, 2, 3, 4, 5, 7, 8, 9, 10, 15, 16
Perill (1)	11, 18,19, 20, 21, 22, 23, 24, 25,26, 27, 28, 29, 30, 31
Obligació (2)	33, 34, 35, 36, 37, 38, 39, 40
Altres (3)	6, 12, 13, 14, 17, 31, 41, 42

Taula 2.3: Relació entre les classes de GTSDB i les seves supercategories

Detectors

Per tal que les xarxes poguessin aprendre a fer la detecció, va caldre reduir el nombre de classes a les mínimes possibles. En primera instància, es va modificar les anotacions de GTSDB agrupant les 43 categories en les supercategories de prohibició, obligació, perill i altres segons la taula 2.3

En aquest experiment, però, no es van utilitzar les mateixes xarxes. Ja no es va entrenar YOLOv4 degut als problemes trobats durant el mètode anterior. A canvi d'aquesta, però, es va afegir la xarxa YOLOv4 Tiny, una versió més lleugera de YOLOv4, que ja no havia estat considerada a la prova anterior per les seves limitacions amb un gran nombre de categories.

Altre cop va caldre modificar les anotacions de GTSDB per tal d'adaptarse a l'entrada de Yolo i modificar la classe `Dataset` de PyTorch per reflectir aquests canvis.

A `traffic_signs/multi_stage/{yolo/faster-rcnn}_training.ipynb` hi ha els notebooks de Jupyter utilitzats per fer aquest entrenament.

A la taula següent es mostren els models entrenats, la mida d'entrada, mAP obtingut i nombre de bilions (i.e. 10^9) d'operacions de punt flotant.

Model	Mida	mAP0.5	BFLOPs
YOLOv4-tiny	1280x736	0.732	36.948
YOLOv3	1280x736	0.941	355.5
Faster R-CNN	1280x736	0.95	1837
SSD MobileNet	1280x736	0.61	2.3

Taula 2.4: Resultats de l'entrenament amb 4 supercatogories

Ja amb més coneixement sobre el funcionament de les xarxes de la família YOLO, es van introduir les següents modificacions:

- **Mides**: La primera i més senzilla va ser la de variar les dimensions de

la capa d'entrada per passar-li imatges de diferent dimensió. Inicialment se li passaven imatges de 1280×736, però també es va provar de passar imatges de mida 640×368, que fa que el nombre de paràmetres de la xarxa es redueixi aproximadament a un quart dels inicials, fent més ràpids tant l'entrenament com la inferència.

- **Anchor boxes**: Tal com ja s'ha explicat a la secció 1.3, les anchor boxes són les caixes per defecte respecte les quals es calculen els desplaçaments de les prediccions. En principi, les seves mides no haurien d'afectar gaire a les prediccions, ja que tota predicció es pot determinar en funció d'una anchor box arbitrària ajustant els desplaçaments. A la pràctica, però, per les xarxes és més senzill aprendre desplaçaments petits i en general s'obté una convergència més ràpida. Per tant, ajustar les mides de les anchor boxes pot ser beneficiós. Per fer-ho, es van recopilar les mides de totes les anotacions de GTSDB i es va utilitzar l'algorisme k-means per trobar les mides més representatives. Aquest algorisme divideix l'espai de mostres en k particions, de manera que cada mostra pertany al clúster amb la mitjana més propera. En el cas de YOLO v3 s'utilitzen 3 capes YOLO amb 3 anchor boxes cada una, obtenint 9 priors diferents, mentre que a YOLO v4 Tiny només s'utilitzen dues capes YOLO amb 3 priors cada una, obtenint 6 caixes diferents. Executant l'algorisme sobre les anotacions del conjunt de dades s'obtenen els clústers amb centres anotats a la taula 2.5.

Capa	**YOLO v3**	**YOLO v4 Tiny**
1a capa YOLO	$(59, 56), (76, 72), (104, 99)$	$(22, 22), (30, 31), (48, 47)$
2a capa YOLO	$(31, 30), (37, 36), (46, 45)$	$(10, 10), (13, 13), (17, 17)$
3a capa YOLO	$(10, 10), (17, 17), (25, 24)$	-

Taula 2.5: Mides de les anchor boxes per cada capa YOLO i imatges de 1280×736

Els resultats obtinguts en aquest experiment es mostren a la taula 2.6 i ja són molt millors que els de l'anterior i comparables amb l'estat de l'art. Com es pot veure, hi ha un augment significatiu de l'mAP, i fins i tot la xarxa YOLOv4-tiny amb la optimització dels anchors obté un resultat superior a YOLOv3 sense optimitzar, però utilitzant només un 10.4% d'operacions. YOLOv3 amb les optimitzacions de les anchors el supera per una mica, però

de cara a utilitzar-la dins el robot, YOLOv4-tiny oferiria molt més rendiment per operació.

Model	**mida**	**mAP@0.5**	**BFLOPs**
YOLOv4-tiny	640x384	0.915	9.64
YOLOv4-tiny	1280x736	0.981	36.97
YOLOv3	640x384	0.942	92.77
YOLOv3	1280x736	0.988	355.6

Taula 2.6: Entrenament amb 4 categories i anchors optimitzades

Preveient que caldria fer un classificador per distingir els senyals detectats per aquesta xarxa, també es va decidir entrenar les xarxes amb una única categoria que contingués tots els senyals. Els resultats obtinguts en aquest entrenament es mostren a la taula 2.7

Model	**mida**	**mAP@0.5**	**BFLOPs**
YOLOv4-tiny	640x384	0.949	9.639
YOLOv4-tiny	1280x736	0.979	36.948
YOLOv3	640x384	0.973	92.740
YOLOv3	1280x736	0.993	355.5
Faster R-CNN Incep. Res 2	1280x736	0.977	1837

Taula 2.7: Entrenament amb una sola classe i anchors optimitzades

El funcionament general de les xarxes amb les 4 supercategories o només amb una és bastant bo quan s'aplica sobre les imatges obtingudes pel robot en entorns reals. Es detecten la majoria de senyals que apareixen i la classificació en supercategories és bona. Les xarxes YOLO no sempre són consistents en imatges consecutives (que d'altra banda tampoc es pot esperar ja que no es passa cap informació referent a les deteccions anteriors), és a dir que en una imatge es fa una detecció correctament però pot ser que a la següent no es faci o canviï de categoria, malgrat aparentment ser molt similar. Faster R-CNN és bastant més consistent, però el cost de fer-hi inferència també és molt més elevat. Un dels majors problemes que comparteixen totes les xarxes és la detecció de falsos positius. Aquests falsos positius inclouen coses com rodes de bicicletes, llums de cotxes, el logo de BMW, cares, rètols amb colors blavosos... Generalment, les probabilitats assignades a la detecció no són gaire altes, però en determinades instàncies se n'han obtingut amb valors de confiança superiors al 80%, de manera que no és suficient augmentar el valor límit

de la probabilitat.

El següent pas ja va ser el de programar els classificadors intentant, a més, reduir el nombre de falsos positius.

Classificadors

La primera idea va ser de fer la classificació mitjançant una altra xarxa neuronal convolucional. Aquesta xarxa només rebria les deteccions fetes per la xarxa detectora, de manera que no hauria de treballar amb imatges tan grans i per tant podria ser una xarxa substancialment més petita. Es va optar per programar una xarxa basada en l'arquitectura d'AlexNet[14], una xarxa àmpliament utilitzada que va ser de les primeres en obtenir bons resultats en tasques de classificació d'objectes amb un gran nombre de classes. A la taula 2.8 es descriu l'arquitectura utilitzada.

Capa	**Mida de sortida**	**Nombre de paràmetres**
Conv2d-1	[-1, 64, 56, 56]	1.792
MaxPool2d-2	[-1, 64, 28, 28]	0
ReLU-3	[-1, 64, 28, 28]	0
Conv2d-4	[-1, 192, 28, 28]	110.784
MaxPool2d-5	[-1, 192, 14, 14]	0
ReLU-6	[-1, 192, 14, 14]	0
Conv2d-7	[-1, 384, 14, 14]	663.936
ReLU-8	[-1, 384, 14, 14]	0
Conv2d-9	[-1, 256, 14, 14]	884.992
ReLU-10	[-1, 256, 14, 14]	0
Conv2d-11	[-1, 256, 14, 14]	590.080
MaxPool2d-12	[-1, 256, 7, 7]	0
ReLU-13	[-1, 256, 7, 7]	0
Dropout-14	[-1, 12544]	0
Linear-15	[-1, 1000]	12.545.000
ReLU-16	[-1, 1000]	0
Dropout-17	[-1, 1000]	0
Linear-18	[-1, 256]	256.256
ReLU-19	[-1, 256]	0
Linear-20	[-1, 43]	11.051

Taula 2.8: Arquitectura de la xarxa utilitzada

Aquesta xarxa es va programar sobre PyTorch[6] i es va entrenar amb el conjunt de dades de GTSRB. Els resultats obtinguts inicialment sobre el conjunt d'avaluació van ser molt bons, arribant a obtenir un **mAP0.5** = 0.961, de manera que es va decidir ajuntar els mòduls de detecció i classificació per avaluar el funcionament global.

Experiments sobre dades reals

En executar tot l'algorisme sobre vídeos de l'ADD, però, els resultats van ser bastant pitjors del que l'mAP havia fet pensar. El mòdul de detecció semblava funcionar prou bé sobre els escenaris reals, amb molt pocs falsos negatius però sí que amb algun fals positiu. Els resultats del classificador, però, ja no eren tan bons. Això va portar a fer un anàlisi de les dades que s'obtenien als experiments i contrastar-les amb les de GTSRB, com a mínim per a comprovar si els conjunts de dades eren més o menys semblants. Les conclusions són les exposades als apartats 2.2.1 i 2.2.1, on s'arriba a la conclusió que tant el context de GTSDB com de GTSRB és substancialment diferent al que s'està exposant el robot. Aquesta diferència no és tan important a la tasca de detecció, ja que, a grans trets, els senyals tenen característiques similars, però es noten molt més a la classificació, on les dades d'entrenament no corresponen a la realitat que es troba.

En aquest punt va ser quan es va decidir crear el nou conjunt de dades a partir de les imatges de l'ADD i el mòdul de detecció.

2.2.4 Mètode 3: Una xarxa neuronal i tècniques tradicionals

L'últim mètode va consistir en fer la classificació mitjançant mètodes més tradicionals de visió i classificació, però mantenint la xarxa encarregada de les deteccions.

En lloc de la xarxa de classificació, l'algorisme proposat extreu informació sobre el color predominant de la senyal i diverses característiques com l'histograma d'orientacions dels gradients i filtres de Leung-Malik[7]. Per a cada color es fa una màquina de vectors de suport que classifica els senyals de la categoria en les classes definides.

[6] La implementació és troba a
`traffic_signs/multi_stage/classifier_training.ipynb`

[7] Veure'n les implementacions a
`traffic_signs/multi_stage/visual-classification.ipynb`

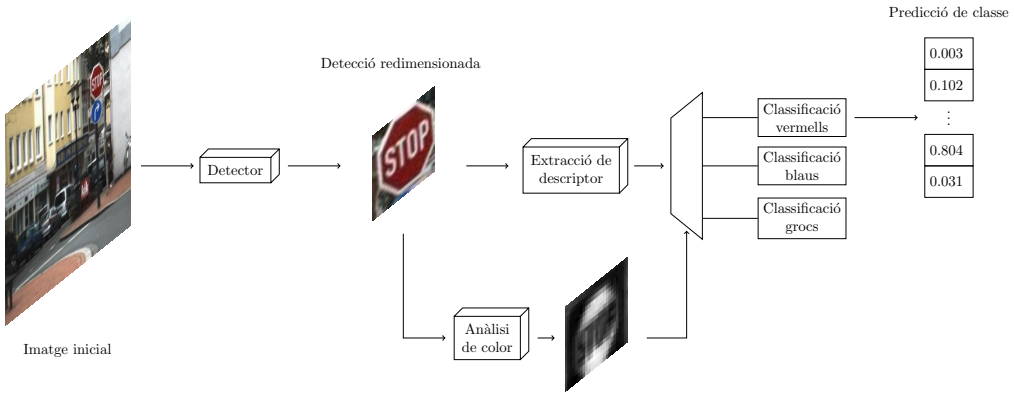


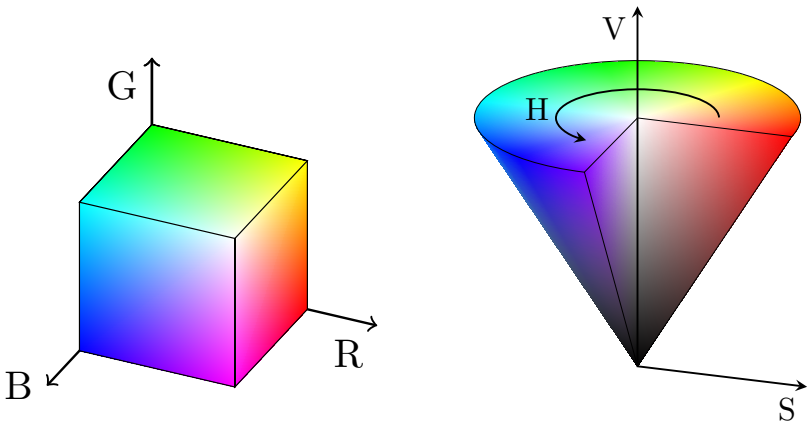
Figura 2.4: Diagrama de funcionament del mètode

Informació de color

El primer pas, doncs, va ser separar els senyals en senyals blaus, vermells i grocs, que són els colors majoritaris que poden contenir. Els senyals blaus corresponen als d'obligació o informació, els vermells als de prohibició i perill i els grocs a casos especials com obres o el senyal de inici i fi de calçada amb prioritat.

Per fer la segmentació per color, es va utilitzar l'espai de representació de color HSV, que es pot veure a la figura 2.5. Aquest espai de color pretén ser més similar a la percepció humana del color modelant com apareixen els colors segons la llum. L'espai es divideix en tres canals diferents: tonalitat (Hue), saturació (Saturation) i valor o brillantor (Value). La informació sobre el color queda determinada pels canals de tonalitat i saturació (el valor de brillantor només indica quanta llum incideix sobre l'objecte), mentre que en sistemes com l'RGB, la informació de color queda distribuïda en els tres canals. Observant la figura 2.5, els avantatges de representació per HSV resulten més clars: és molt més senzill delimitar la secció del con que correspon al color que busquem que determinar l'espai en el cub RGB corresponent al mateix color.

Malgrat tot, establir els valors límits sempre és complicat: massa estrictes deixen algun espectre fora; massa laxos permeten que colors que no volem detectar també hi entrin. Per això, sovint s'aplica alguna transformació als valors de color per tal de facilitar aquesta feina. A [6] i [8] fan servir taules de valors que assignen valors alts al color que es vol detectar i a mesura que es van allunyant van decreixent de manera lineal fins arribar a 0. A la figura 2.6 es veu l'exemple de transformació que es fa a [8] pel color blau, on s'utilitzen tres



(a) (b)

Figura 2.5: Representacions de l'espai RGB (a) i HSV (b)

paràmetres: els valors hb_{min} i hb_{max} el primer i últim valor en què es considera que el color és blau i hb_{top}, el valor corresponent a la tonalitat de blau mitjana.

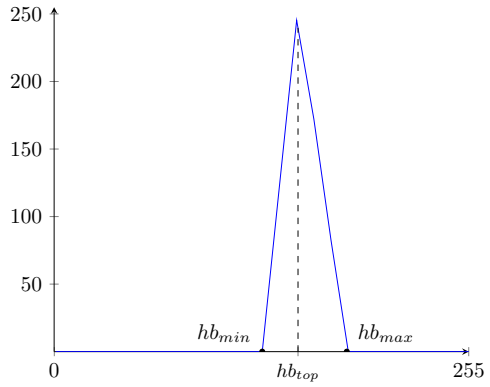


Figura 2.6: Transformació pel color blau. A l'eix vertical hi ha els valors de píxel inicial, a l'eix horitzontal els valors transformats.

En aquest cas concret, també s'ha utilitzat una transformació de colors, però en lloc de fer-ho per taules de valors es fa mitjançant distribucions gaussianes. Aquestes gaussianes s'apliquen tant al canal de tonalitat com al de saturació i es genera una nova imatge a partir del producte dels dos valors. Les funcions utilitzades són:

$$h_{blau}(x) = exp\left(\frac{-(x-170)^2}{30^2}\right)$$

$$h_{vermell}(x) = exp\left(\frac{-x^2}{20^2}\right) + exp\left(\frac{-(x-255)^2}{20^2}\right)$$
$$h_{groc}(x) = exp\left(\frac{-(x-20)^2}{20^2}\right)$$
$$hs(x) = exp\left(\frac{-(x-255)^2}{115^2}\right)$$

Les funcions $h_i \colon [0, 255] \to [0, 1]$ amb $i \in \{\text{blau}, \text{vermell}, \text{groc}\}$ reben els valors del canal de tonalitat i $hs \colon [0, 255] \to [0, 1]$ els valors de saturació. Cada gaussiana està centrada en el valor més pur corresponent al seu color i la variància està ajustada per cobrir l'espectre de tonalitat desitjat. Aleshores, per a cada imatge es calcula $h_i \times hs$, $i \in \{\text{blau}, \text{vermell}, \text{groc}\}$ píxel a píxel, obtenint una nova imatge d'un sol canal amb valors entre 0 i 1, que en certa manera actuen com a probabilitat de que cada píxel sigui o no d'un color. A la figura 2.7 es pot veure un exemple de càlcul sobre una imatge real.

Figura 2.7: D'esquerra a dreta: imatge en RGB, canal H, canal S, activació $h_{vermell}$, activació hs, producte $h_{vermell} \times hs$

Figura 2.8: D'esquerra a dreta: imatge en RGB, canal H, canal S, activació h_{blau}, activació hs, producte $h_{blau} \times hs$

Un cop s'han obtingut les tres imatges, es posen a 0 tots els píxels que no superen un cert llindar (0.25 en el nostre cas) i es calcula l'àrea total dels píxels restants. Aleshores, un com fet aquest procés pels tres colors, es classifica cada senyal en funció del color amb àrea més gran.

Extracció de característiques

La característica més rellevant en senyals a part del color és la forma, així que la idea inicial va ser fer una classificació més fina a partir de color i forma.

Per fer-ho, s'agafa el mapa de probabilitat amb àrea màxima del pas anterior, es divideix en n punts i es calcula la distància entre el punt del marge i el primer píxel no aïllat amb valor superior a 0 o, de manera equivalent, igual al llindar establert en el pas anterior. Les distàncies obtingudes per a cada costat es normalitzen i es concatenen, obtenint així un descriptor de la forma. Els resultats obtinguts amb aquest descriptor són suficients per determinar la forma, però pressuposa que la capsa contenidora del senyal que dóna el detector és el més ajustada possible. Això, però, sovint no és cert. Per les imatges on la capsa contenidora és més gran, el problema es va solucionar trobant una capsa contenidora més ajustada reduint les dimensions fins a trobar algun píxel amb valor superior a 0. El problema, però, sorgeix amb les capses contenidores que són més petites que el senyal, on la imatge passada al classificador ja no té tota la informació. En aquest cas, s'hauria de tornar a la imatge inicial passada al detector i extreure'n allà la capsa contenidora correcta.

La següent opció és analitzar directament el contingut de cada senyal a partir dels pictogrames que contenen. Per fer-ho, es genera un descriptor a partir del HOG (histograma d'orientacions del gradient) i un conjunt de filtres de Leung-Malik. Els filtres de Lenung-Malik es calculen a partir de desplaçaments, diferències i laplacians de gaussianes amb diferents escales i orientacions. Aquests filtres s'apliquen a les imatges i, per a cada filtre, es pren la mitjana dels valors absoluts de les activacions. Concatenant els valors del HOG i els obtinguts a partir dels filtres de Leung-Malik, es genera el descriptor de la imatge.

Un cop definit el descriptor ja es pot classificar la imatge en les categories, que en aquest cas s'ha realitzat a partir de màquines de vectors de suport. Així, s'ha generat tres màquines de vectors de suport, una per a cada color predominant.

Per entrenar-les, havent vist els mals resultats obtinguts sobre proves reals amb xarxes entrenades amb GTSRB, es va fer servir el nou conjunt de dades generat.

A aquestes categories es va afegir la categoria `no_senyal`, que conté exemples de falsos positius trobats pel detector. A la pràctica, es va observar que molts d'aquests falsos positius ja queden filtrats per la classificació preliminar en colors on, si no hi ha un mínim d'activació per cap color, es descarta la detecció. Malgrat tot, s'ha afegit aquesta classe per intentar captar els falsos positius que passen aquest filtre. Realment no s'espera que aquesta classe tingui bona precisió ni exhaustivitat, ja que el conjunt de no senyals té variància infinita i és impossible representar-lo amb les mostres de falses deteccions que conté el conjunt d'entrenament.

Pel que fa als colors representatius de les senyals, les classes de prohibició d'aparcament i prohibició d'aparcament i estacionament es classifiquen tant a la SMV vermella com a la blava. Inicialment aquests senyal només eren a la blava, però es va veure que a vegades es detectaven com a vermelles, i afegint-les a les dues es maximitza la possibilitat de detectar-les correctament.

Una altra característica interessant és que hi ha senyals diferents que tenen descriptors molt similars. Per exemple, els senyals d'obligació d'anar cap a l'esquerra i cap a la dreta tenen exactament els mateixos gradients i activacions dels filtres molt similars, de manera que és molt complicat separar-les i s'ajunten dins una mateixa classe. Això a la pràctica no suposa un problema gaire gran ja que, per la pròpia naturalesa dels senyals, és poc probable apareguin alhora en una mateixa imatge.

Fent la correlació de les classes seleccionades amb les de GTSDB es pot veure que només hi apareixen 15 de les 27 que s'han considerat com a més representatives.

Experiments sobre dades reals

Un cop construït el conjunt de dades, només queda entrenar les tres màquines de suport de vectors. A la taula 2.9 es pot veure els resultats de l'avaluació i a les figures 2.9, 2.11 i 2.10 les matrius de confusió de l'SVM blava, vermella i groga respectivament.

En executar-ho sobre les imatges de l'ADD, es pot veure que els resultats prou bons. L'algorisme de detecció és el mateix que al mètode 2, però afegint el filtre de color es redueixen dràsticament els falsos positius, fent que pràcticament totes les deteccions siguin realment de senyals de trànsit. S'ha observat classificacions errònies sobretot entre els senyals d'obligació que contenen fletxes i entre algunes de prohibició, on la classificació oscil·la entre diverses classes. Malgrat tot, és molt estable en els senyals clarament diferenciats com stop, prohibit circular, prohibit passar, cediu el pas, etc.

SVM	**Precisió**	**Exhaustivitat**	**F1**
Blau	0.977	0.973	0.975
Vermell	0.944	0.979	0.953
Groc	0.881	0.984	0.976

Taula 2.9: Avaluació de les tres SVM

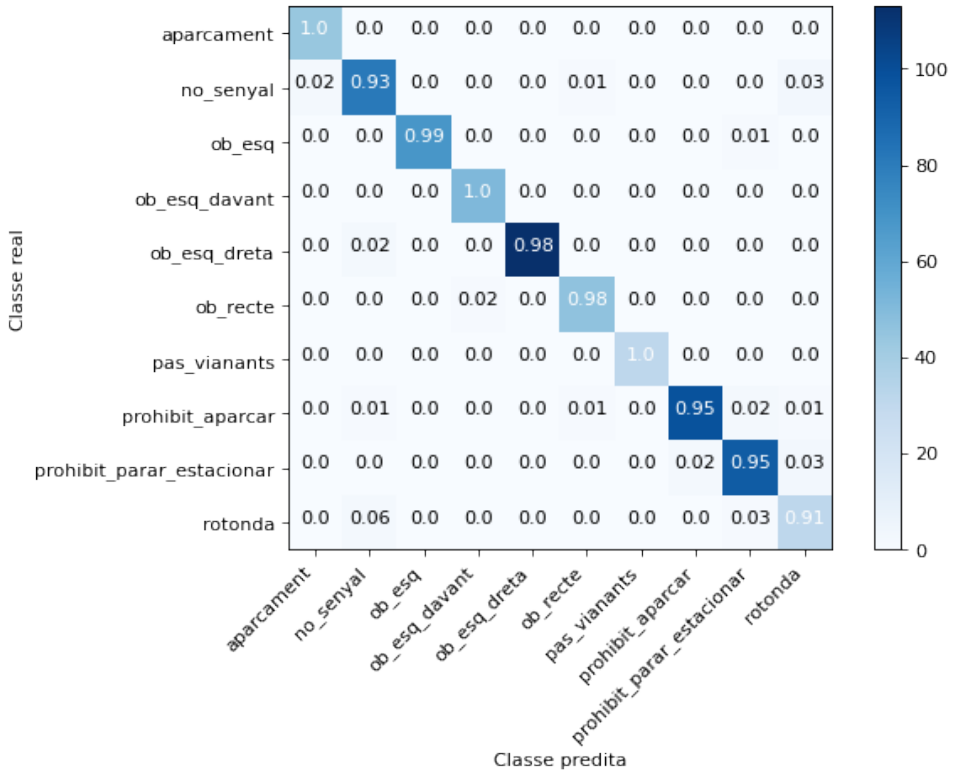


Figura 2.9: Matriu de confusió dels senyals blaus

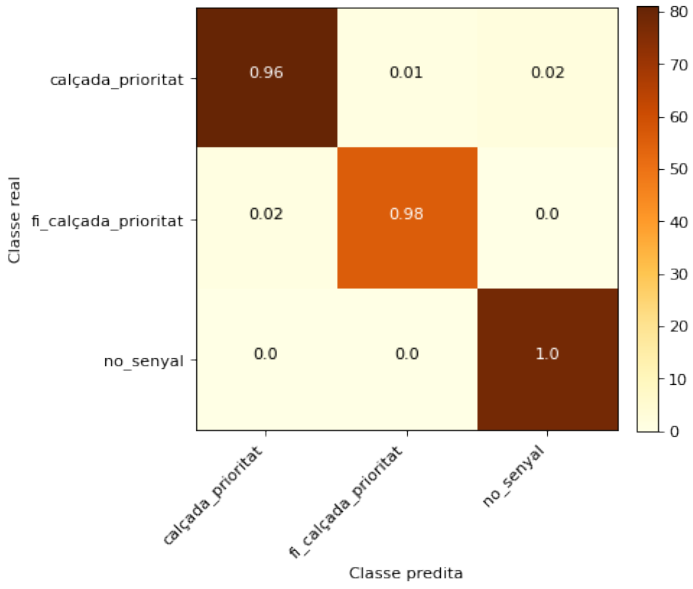


Figura 2.10: Matriu de confusió dels senyals grocs

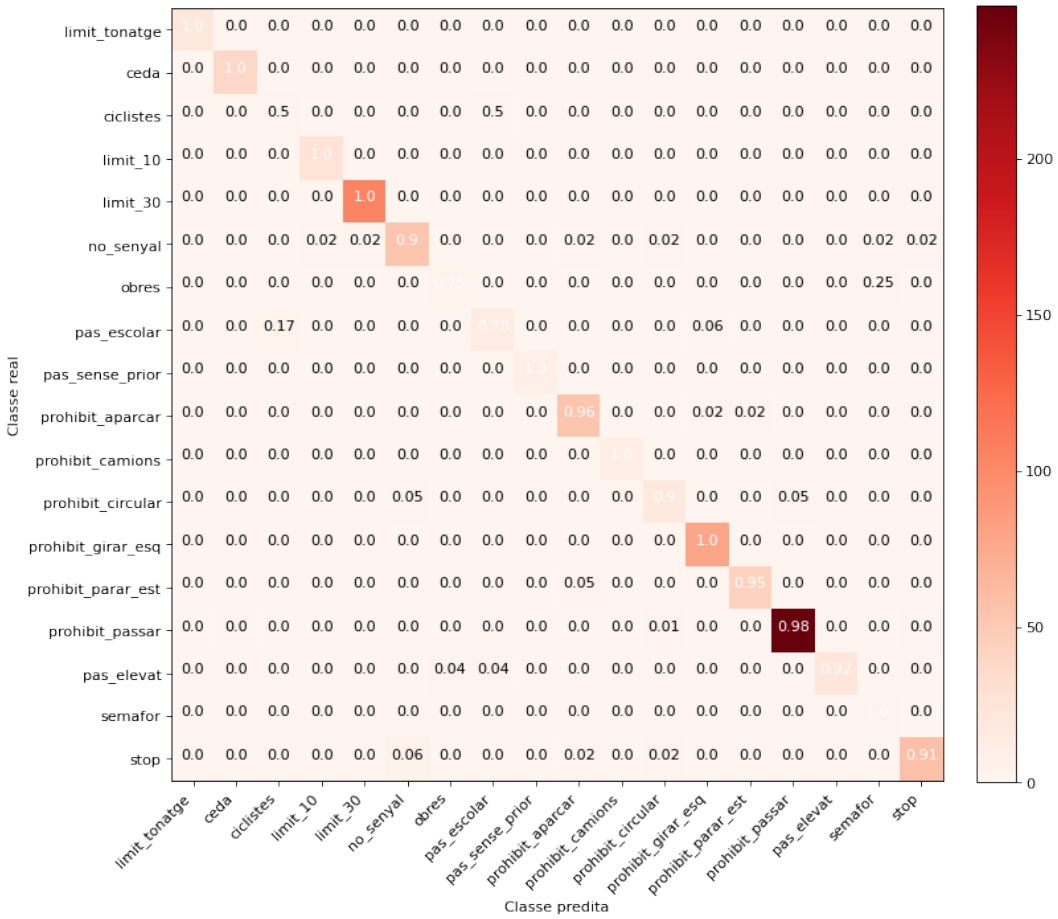


Figura 2.11: Matriu de confusió dels senyals vermells

Capítol 3

Conclusions

Durant aquests mesos s'ha anat treballant per assolir els objectius proposats inicialment. Entre ells hi havia l'objectiu d'entendre els funcionament de les eines utilitzades, que es desenvolupa majoritàriament dins el capítol 1 d'aquesta memòria. Aquest coneixement teòric ha estat especialment útil en el cas de les xarxes YOLO, on s'ha pogut trobar la funció de pèrdua de la versió 3 i s'ha pogut optimitzar les xarxes obtenint-ne millors resultats.

Pel que fa als objectius més pràctics, s'ha aconseguit desenvolupar un detector que obté molts bons resultats, tant en el dataset GTSDB com en vídeos directes de l'ADD, al nivell dels mètodes de l'estat de l'art. L'objectiu de desenvolupar el classificador de senyals no ha estat tan satisfactori, però. Malgrat que els models obtenen bons resultats a nivell teòric, quan s'executen en entorns reals els resultats ja no són tan bons. Això és atribuïble a la diferència de context entre les dades d'entrenament i les que es troben a la realitat i, malgrat que s'ha intentat solucionar creant un conjunt de dades amb més similitud a l'entorn, ni els models ni les dades han estat prou bons per obtenir el nivell de robustesa desitjat.

Aquesta disparitat entre el funcionament sobre els conjunts de dades de senyals alemanys i sobre escenaris reals ha estat un dels majors obstacles durant el treball. Això també em fa qüestionar si els articles estudiats realment estan resolent el problema de la detecció i classificació en general, o bé només optimitzant els models per a aquestes dades.

3.1 Treball Futur

Per manca de temps i la dificultat logística i administrativa que requereix treure el robot del laboratori i fer proves al carrer, no ha estat possible validar els mètodes amb el robot real, de manera que aquesta seria la primera tasca pendent.

Per millorar els resultats de la classificació caldria un conjunt de dades més representatiu, més gran i més equilibrat. El conjunt d'imatges que s'ha creat és un pas en la bona direcció, però no és ni molt menys òptim. Seria necessari afegir més variabilitat d'imatges dins de cada classe en lloc de dependre només de l'augmentació de dades. Caldria, també, tenir una distribució d'imatges per classe més balancejada.

A més, durant el projecte també s'ha pensat diferents millores:

- Tenir en compte el component temporal. Actualment el sistema treballa frame a frame, però tenint en compte que són consecutius, es podria implementar un sistema que en fes el seguiment i que utilitzés múltiples imatges per fer la predicció. Això comportaria un retard a la detecció inicial, però podria evitar problemes com la oscil·lació de la predicció de la classe o les oclusions parcials.

- Un millor sistema de filtratge per a les deteccions fora de distribució. Això s'ha aconseguit en certa manera mitjançant el filtre de color al mètode 3, i certament les últimes xarxes de detecció amb les modificacions ja eren prou bones, però algun fals positiu encara el passa.

- Sistemes de classificació alternatius. Essent la classificació la part més feble de l'algorisme, s'ha plantejat múltiples mètodes alternatius per realitzar-la, com l'extracció i anàlisi de pictogrames, l'ús d'OCR, descriptors més geomètrics o basats en simetries... Alguns d'aquests mètodes s'han començat a implementar al notebook `traffic_signs/multi_stage/visual-classification.ipynb`.

En una escala més gran, també s'ha plantejat la detecció de punts de referència alternatius. En aquest sentit es va començar a desenvolupar dos mètodes alternatius:

- La detecció de punts de referència en el pla del terra, com podrien ser tapes de claveguera, senyalització horitzontal o línies de carril. Per fer-ho s'utilitza una projectivitat que canvia el sistema de referència de la

càmera, posant-lo ortogonal al pla del terra. La implementació inicial es troba a `birds_eye/`.

- La detecció del punt de fuga de la imatge. Es pot calcular fent optimització de mínims quadrats per la intersecció de múltiples línies rectes detectades a partir de la transformada de Houg. Trobant aquest punt, coneixent els paràmetres intrínsecs de la càmera i la seva posició al robot es pot obtenir una estimació de la variació de la orientació del robot, una estimació del pla del terra i se simplifica la detecció de línies de carril entre d'altres. La implementació actual es troba a `vanishing_point/`.

Referències

[1] Álvaro Arcos-García, Juan A. Álvarez-García i Luis M. Soria-Morillo. "Evaluation of deep neural networks for traffic sign detection systems". A: *Neurocomputing* 316 (2018), pàg. 332 - 344. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2018.08.009. URL: https://www.sciencedirect.com/science/article/pii/S092523121830924X.

[2] Nick Barnes i Alex Zelinsky. "Real-time radial symmetry for speed sign detection". A: (2004), pàg. 566 - 571.

[3] S Maldonado Bascón et al. "An optimization on pictogram identification for the road-sign recognition task using SVMs". A: *Computer Vision and Image Understanding* 114.3 (2010), pàg. 373 - 383.

[4] Alexey Bochkovskiy, Chien-Yao Wang i Hong-Yuan Mark Liao. "YOLOv4: Optimal Speed and Accuracy of Object Detection". A: *CoRR* abs/2004.10934 (2020). arXiv: 2004.10934. URL: https://arxiv.org/abs/2004.10934.

[5] O. Calin. *Deep Learning Architectures: A Mathematical Approach*. Springer Series in the Data Sciences. Springer International Publishing, 2020. ISBN: 9783030367206. URL: https://books.google.es/books?id=KXtdywEACAAJ.

[6] Arturo De la Escalera, J Ma Armingol i Mario Mata. "Traffic sign recognition and analysis for intelligent vehicles". A: *Image and vision computing* 21.3 (2003), pàg. 247 - 258.

[7] A. de la Escalera et al. "Road traffic sign detection and classification". A: *IEEE Trans. Ind. Electron.* 44.6 (des. de 1997), pàg. 848 - 859.

[8] Hilario Gómez-Moreno et al. "Goal evaluation of segmentation algorithms for traffic sign recognition". A: *IEEE Transactions on Intelligent Transportation Systems* 11.4 (2010), pàg. 917 - 930.

[9] Yang Gu i Bingfeng Si. "A novel lightweight real-time traffic sign detection integration framework based on YOLOv4". A: *Entropy* 24.4 (2022), pàg. 487.

[10] Sebastian Houben et al. “Detection of Traffic Signs in Real-World Images: The German Traffic Sign Detection Benchmark”. A: 1288 (2013).

[11] Sergey Ioffe i Christian Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. A: *International conference on machine learning*. PMLR. 2015, pàg. 448 - 456.

[12] H. Kamada, S. Naoi i T. Gotoh. “A compact navigation system using image processing and fuzzy control”. A: *Proc. Southeastcon* 1 (abr. de 1990), pàg. 337 - 342.

[13] D.S. Kang, N.C. Griswold i N. Kehtarnavaz. “An invariant traffic sign recognition system based on sequential color processing and geometrical transformation”. A: *IEEESouthwest Symposium on Image Anal ysis and Interpretation, IEEE* (abr. de 1994).

[14] Alex Krizhevsky, Ilya Sutskever i Geoffrey E. Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. A: *Commun. ACM* 60.6 (maig de 2017), pàg. 84 - 90. ISSN: 0001-0782. DOI: 10.1145/3065386. URL: https://doi.org/10.1145/3065386.

[15] Jia Li i Zengfu Wang. “Real-time traffic sign recognition based on efficient CNNs in the wild”. A: *IEEE Transactions on Intelligent Transportation Systems* 20.3 (2018), pàg. 975 - 984.

[16] Carlos Filipe Paulo i Paulo Lobato Correia. “Automatic detection and classification of traffic signs”. A: (2007), pàg. 11 - 11.

[17] Paloma de la Puente i Diego Rodrıguez-Losada. “Feature based graph-SLAM in structured environments”. A: *Autonomous Robots* 37.3 (2014), pàg. 243 - 260.

[18] Joseph Redmon i Ali Farhadi. “YOLO9000: Better, Faster, Stronger”. A: *CoRR* abs/1612.08242 (2016). arXiv: 1612.08242. URL: http://arxiv.org/abs/1612.08242.

[19] Joseph Redmon i Ali Farhadi. “YOLOv3: An Incremental Improvement”. A: *CoRR* abs/1804.02767 (2018). arXiv: 1804.02767. URL: http://arxiv.org/abs/1804.02767.

[20] Joseph Redmon et al. “You Only Look Once: Unified, Real-Time Object Detection”. A: *CoRR* abs/1506.02640 (2015). arXiv: 1506.02640. URL: http://arxiv.org/abs/1506.02640.

[21] G.K. Siogkas i E.S. Dermatas. “Detection, Tracking and Classification of Road Signs in Adverse Conditions”. A: (maig de 2006), pàg. 537 - 540.

[22] Johannes Stallkamp et al. “The German Traffic Sign Recognition Benchmark: A multi-class classification competition”. A: *IEEE International Joint Conference on Neural Networks*. 2011, pàg. 1453 - 1460.

[23] Domen Tabernik i Danijel Skočaj. “Deep learning for large-scale traffic-sign detection and recognition”. A: *IEEE transactions on intelligent transportation systems* 21.4 (2019), pàg. 1427 - 1440.

[24] Sebastian Thrun i Michael Montemerlo. “The Graph SLAM Algorithm with Applications to Large-Scale Mapping of Urban Structures”. A: *The International Journal of Robotics Research* 25.5-6 (2006), pàg. 403 - 429. DOI: 10.1177/0278364906065387.

[25] Yi Yang et al. “Towards real-time traffic sign detection and classification”. A: *IEEE Transactions on Intelligent transportation systems* 17.7 (2015), pàg. 2022 - 2031.