

Long-Term Visual Object Tracking

Visió per Computador
Short Project

Joan Caballero, Oriol Miró
Facultat d'Informàtica de Barcelona
Curs 22-23 Q2

Índex

1	Introducció	3
1.1	Descripció de la base de dades	3
2	Tracking	4
2.1	Mètode Propi	4
2.1.1	Descripció del mètode	4
2.1.2	Estudi del detector de keypoints	5
2.1.3	Problemes trobats	6
2.1.3.1	Pèrdua de l'objecte durant el tracking	6
2.1.3.2	Matching de keypoints que no pertanyen a l'objecte	7
2.1.3.3	Superposició d'objectes que provoca un canvi d'objectiu	7
2.1.4	Funcions utilitzades	7
2.2	Mètode No Propi	8
2.2.1	Descripció del mètode	8
2.2.2	Problemes trobats	9
2.2.2.1	Pèrdua de l'objecte durant el tracking	10
2.2.3	Funcions utilitzades	10
3	Reconeixement	12
3.1	Mètode Propi	12
3.1.1	Descripció del mètode	12
3.1.2	Etapes del treball realitzat	12
3.1.2.1	Entrenament del classificador	12
3.1.2.2	Predictions sobre les imatges	13
3.1.2.3	Càcul de la precisió	14
3.1.3	Estudi del descriptor i el classificador	14
3.1.3.1	Resultats del entrenament	15
3.1.4	Problemes trobats	18
3.1.4.1	Realitzar <i>feature engineering</i>	18
3.1.4.2	Triar una finestra inicial	18
3.1.4.3	Zooms en les imatges	19
3.1.5	Funcions utilitzades	19
3.2	Mètode No Propi	20
3.2.1	Descripció del mètode	20
3.2.2	Etapes del treball fet	20
3.2.2.1	Preparació de les dades	20
3.2.2.2	Entrenament del model	21
3.2.2.3	Càcul de la precisió	21
3.2.3	Problemes trobats	21
3.2.4	Funcions utilitzades	22

4 Resultats	24
4.1 Resultats Tracking Mètode Propi	24
4.1.1 Anàlisi	24
4.2 Resultats Tracking Mètode No Propi	27
4.2.1 Anàlisi	27
4.3 Resultats Reconeixement Mètode Propi	29
4.3.1 Anàlisi	29
4.4 Resultats Reconeixement Mètode No Propi	31
4.4.1 Anàlisi	31
5 Conclusió	32
A Annex A: Codi	34
A.1 Codi Tracking Mètode Propi	34
A.2 Codi Tracking Mètode No Propi	39
A.3 Codi Reconeixement Mètode Propi	43
A.4 Codi Reconeixement Mètode No Propi	52
B Entrenament dels classificadors	57
B.1 Mètode Propi	57
B.2 Mètode No Propi	65

1 Introducció

En aquest informe abordarem la tasca proposada pel *Short Project*, de l'assig-natura de Visió per Computador. La tasca consisteix en implementar mètodes per realitzar el seguiment d'objectes en seqüències d'imatges utilitzant la base de dades *TinyTLP*.

Específicament, ens demanen implementar un mètode propi i un mètode no propi per a cada tipus de mètode. Un mètode propi és aquell que ha estat concebut i dissenyat per nosaltres mateixos, mentre que un mètode no propi és aquell que utilitza una llibreria o solució extreta d'Internet o altres fonts externes.

1.1 Descripció de la base de dades

Segons la pàgina web, el conjunt de dades *TLP* consisteix en 50 seqüències HD llargues, amb un total de 676.431 frames. Cada seqüència està formada per un únic objecte a seguir, marcat en el primer frame. D'altra banda, *TinyTLP* és una versió de curta durada derivada de *TLP*, que conté els primers 600 frames (20 segons) de cada seqüència del conjunt de dades *TLP* [3].

En aquest projecte, farem servir 15 seqüències d'aquest conjunt de dades per provar els nostres mètodes. Les seqüències seleccionades són les següents: *Alladin*, *Bike*, *Boat*, *CarChase1*, *Dashcam*, *Driftcar2*, *Drone1*, *Drone2*, *Drone3*, *Jet3*, *Mohiniyattam*, *MotorcycleChase*, *PolarBear1*, *Puppies1* i *Violinist*.

Hem triat aquestes seqüències perquè contenen tots els tipus de dificultats possibles, com ara oclusions, fons poc diferenciables, moviments ràpids, canvis d'escala i variacions d'il·luminació.

Per desenvolupar els nostres models sense afectar excessivament els resultats, utilitzarem només tres d'aquestes seqüències que representin una varietat de desafiaments i dificultats creixents:

- *MotorcycleChase*
- *Bike*
- *Violinist*

2 Tracking

En aquesta primera part se'ns demana implementar un sistema de seguiment d'un objecte, animal o persona en una seqüència d'imatges mitjançant tècniques de tracking. A partir de la posició de l'objecte en un frame determinat, hem de detectar on estarà situat l'objecte en el següent frame.

Per a la tècnica de tracking no se'ns permet utilitzar la posició del quadre delimitador real de l'objecte, però sí que podem utilitzar la mida real del quadre delimitador per facilitar el procediment de tracking. Aquesta informació ens la proporciona el fitxer *groundtruth_rect.txt* que ve adjunt amb la seqüència d'imatges.

2.1 Mètode Propi

En aquest apartat explicarem com hem implementat el mètode propi per tal de fer el nostre tracking d'objectes.

Per implementar aquesta tècnica ens hem basat en els continguts que hem vist durant tot el curs en l'assignatura de Visió per Computador. Veurem que aplicarem coneixements de descriptors d'imatges, extractors de característiques i procediments de MATLAB que hem vist durant el transcurs de l'assignatura.

Cal mencionar que el codi d'aquest mètode està totalment implementat per nosaltres amb les idees que se'ns han anat ocorrent durant el desenvolupament del projecte.

2.1.1 Descripció del mètode

El mètode propi que hem desenvolupat consisteix a detectar certes característiques importants (keypoints) de l'objecte a seguir en un frame per tal de trobar-les en el següent frame.

Cerquem els keypoints de l'objecte en el frame actual en les proximitats de la posició de l'objecte en el frame anterior, i fem matching dels keypoints. Anem repetint aquest procés per a cada frame fins que perdem el tracking de l'objecte o s'acaben els frames.

Per escollir quin tipus de característiques havíem d'extreure dels objectes hem testejat amb diferents mètodes de detecció de keypoints, hem realitzat un estudi comparant el rendiment de cadascun d'ells i hem seleccionat el mètode que millor rendiment ha obtingut en aquest estudi.

2.1.2 Estudi del detector de keypoints

Per l'estudi hem decidit seleccionar seqüències d'imatges de la base de dades de TinyTLP que tingessin diferents característiques entre elles: en algunes es troben dificultats per detectar l'objecte a causa del nivell d'il·luminació, en altres l'objecte fa moviments bruscs i es pot perdre amb facilitat, i en alguns l'objecte canvia bastant d'aparença dificultant la detecció de característiques importants. Les seqüències d'imatges usades han estat: *MotorcycleChase* (moviments bruscs), *Bike* (canvi d'aparença i moviments bruscs) i *Violinist* (il·luminació).

A continuació es mostra la taula indicant, per a cada mètode i seqüència d'imatges, el percentatge d'encert i el temps d'execució. Per a aquest estudi hem representat el percentatge d'encert com el percentatge de frames on el quadre delimitador que troba el mètode i el quadre delimitador real proporcionat per *groundtruth_rect.txt* es superposen amb més d'un 60%.

Mètode	MotorcycleChase		Bike		Violinist	
	%	Temps (s)	%	Temps (s)	%	Temps (s)
SIFT	99	31,4	52	16,57	23	19,66
SURF	100	18,92	1	17,35	5	15,99
Harris	98	23,07	21	44,17	22	45,73
FAST	89	17,03	41	35,5	10	31
MinEigen	82	23,29	9	45,53	20	41,72
BRISK	88	75,44	32	149,66	24	153,94

Taula 1: Estudi dels detectors de característiques per tracking

Hem ordenat les seqüències d'imatges creixentment segons la seva dificultat per fer tracking, sent *MotorcycleChase* la més senzilla i *Violinist* la més complicada. Cal mencionar que hi ha mètodes que tenen un percentatge molt baix d'encert en algunes seqüències d'imatges, i això és degut a que el mètode ha perdut totalment l'objecte en algun punt de la seqüència i per tant en tots els següents frames no detecta l'objecte.

Això és degut a que per aquest estudi no hem implementat el filtre de Kalman per predir la posició de l'objecte en cas que es perdés. Volem que els resultats obtinguts siguin exclusivament producte del mètode de detecció de característiques triat, i no volem que el filtre de Kalman alteri les dades, per això no l'hem posat.

A continuació es mostra un resum de la taula anterior, on es mostra per a cada descriptor, la mitjana del percentatge d'encerts i la mitjana del temps esperat per a les tres seqüències d'imatges.

Mètode	Precisió (%)	Temps (s)
SIFT	58,00	22,54
BRISK	48,00	126,35
FAST	46,67	27,84
Harris	47,00	37,66
MinEigen	37,00	36,85
SURF	35,33	17,42

Taula 2: Conclusió de l'estudi dels detectors de característiques

Com podem veure el descriptor que millor rendiment ha obtingut és SIFT (Scale-invariant feature transform), tant pel seu alt percentatge d'encerts com pel seu ràpid temps d'execució. Aquest és el motiu pel qual hem decidit usar SIFT com el nostre descriptor de tracking a l'hora d'avaluar els resultats.

2.1.3 Problemes trobats

A continuació presentem una sèrie de problemes que hem enfrontat durant la implementació de l'algorisme de tracking. Per a cadascun d'ells explicarem la situació on produeixen aquests problemes i com els hem intentat solucionar.

És important remarcar que, habitualment, els mètodes de tracking es complementen amb mètodes de detecció que ajuden al tracking quan es perd de vista l'objecte. No obstant això, el nostre algorisme es basa exclusivament en tècniques de tracking i no es combina amb cap altre mètode. A causa d'això, ens trobem amb diverses limitacions en alguns aspectes.

2.1.3.1 Pèrdua de l'objecte durant el tracking

En el nostre algorisme, ens trobem amb situacions en què detectem els keypoints de l'objecte en el frame actual i no aconseguim fer matching amb els keypoints del frame anterior. Això pot ser a causa que no som capaços de detectar keypoints mitjançant SIFT o perquè, tot i què detectem keypoints, aquests no coincideixen amb els del frame anterior i no podem fer matching entre ells.

Per abordar aquest problema, vam implementar una versió del codi on ens ajudem del predictor de la posició del quadre delimitador de l'objecte utilitzant el filtre de Kalman. No obstant això, els resultats obtinguts no van ser satisfactoris. Kalman no predeia correctament les noves posicions i provocava que els quadres delimitadors es desviessin de l'objecte, ja que es pensava que l'objecte aniria en una direcció quan, en realitat, avançava cap a una altra. Aquest problema ens va portar a descartar aquesta possibilitat com a solució.

Una altra possibilitat era no buscar l'objecte en les proximitats del quadre delimitador del frame anterior, sinó en tota la imatge. En canvi, a més de ser això una tècnica més de detecció que de tracking, tampoc va funcionar bé, ja que

es detectaven keypoints que no pertanyien a l'objecte i, per tant, el seguiment fallava.

Per tant, davant del fracàs d'aquestes dues alternatives, vam decidir que, en situacions com aquestes, assignaríem el quadre delimitador del frame actual a la mateixa posició que tenia el quadre delimitador del frame anterior.

2.1.3.2 Matching de keypoints que no pertanyen a l'objecte

En algunes situacions, la detecció per SIFT extrau punts característics del fons de la imatge que no pertanyen a l'objecte, provocant que es facin matching de keypoints que no són rellevants per a l'objecte.

Això pot ocasionar desplaçaments del quadre delimitador, que originalment estava centrat en l'objecte, en una direcció errònia a causa d'aquests keypoints "outliers", cosa que provoca que el nou quadre delimitador ja no estigui centrat en l'objecte i que per tant es vagi perdent el seu tracking poc a poc.

Per evitar això hem usat la funció 'selectStrongest(x)' que proporciona la llibreria de Computer Vision Toolbox de MATLAB. Aquesta funció ens permet seleccionar els x keypoints més rellevants. Hem realitzat diversos experiments utilitzant diferents valors per a aquesta variable x en diferents vídeos, i finalment hem determinat que un valor de 10 és el que funciona millor per al nostre algorisme.

Això significa que seleccionem els deu punts clau més característics per garantir una millor correspondència i evitar l'afectació de punts clau no desitjats en el seguiment de l'objecte.

2.1.3.3 Superposició d'objectes que provoca un canvi d'objectiu

En certes situacions, pot ocórrer que un segon objecte es col·loqui davant de la càmera, i el detector SIFT pugui detectar keypoints d'aquest nou objecte. Això pot provocar que, gradualment, el seguiment es desplaci cap a la figura del segon objecte i, finalment, l'objectiu del tracking es canviï completament.

Desafortunadament, no hem aconseguit solucionar aquest problema, i existeixen vídeos on aquesta situació fa que el nostre sistema de seguiment falli. Aquesta és una limitació important que hem intentat abordar per millorar la robustesa i l'exactitud del nostre algorisme de tracking, però no hem pogut trobar una solució utilitzant les tècniques que hem après en l'assignatura.

2.1.4 Funcions utilitzades

Pel mètode de tracking propi hem implementat una sèrie de funcions que ens ajuden a modularitzar el codi i fer-lo més entenedor:

- **getRectanglePercentage(rect, perc):** Aquesta funció pren com a paràmetres un rectangle i un percentatge i retorna el mateix rectangle incrementat segons el percentatge passat com a paràmetre.

Utilitzem aquesta funció per a buscar l'objecte en una zona propera d'on es troava l'objecte en el frame anterior.

- **resizeRectangle(rect, width, height):** Aquesta funció rep un rectangle, una amplada i una alçada i retorna el rectangle redimensionat amb les mides passades com a paràmetre.

Utilitzem aquesta funció per redimensionar el quadre delimitador obtingut per tracking segons les mides que ens proporciona el predictor del filtre de Kalman.

- **rectangleFromKP(kp_coordinates, rect):** Aquesta funció rep com a paràmetres les posicions d'uns keypoints i un rectangle i retorna un quadre delimitador que engloba tots els keypoints ajustat al rectangle passat per paràmetre.

Ens és útil per a trobar el quadre delimitador que indica la posició de l'objecte a partir dels keypoints.

- **correctRectanglePos(prev_rect, curr_rect):** Aquesta funció pren com a paràmetres el quadre delimitador de l'objecte en el frame anterior i en el frame actual, i retorna el quadre delimitador del frame actual ajustat amb la posició del quadre delimitador del frame anterior.

Ens serveix per calcular la posició del quadre delimitador del frame actual quan aquest es separa bastant del quadre delimitador anterior. Quan el mètode de tracking troba dos quadres delimitadors consecutius molt separats entre ells, amb aquesta funció disminuïm la distància entre els dos per a què quadres delimitadors consecutius estiguin a prop un de l'altre.

2.2 Mètode No Propi

En aquesta secció explicarem com s'ha realitzat el mètode no propi per a tracking. L'algorisme que hem utilitzat i tota la informació relacionada amb ell s'ha extret del paper Real-Time Compressive Tracking. També es pot accedir a la seva pàgina web on es mostra més informació i es troba el codi que hem utilitzat mitjançant aquest enllaç.

2.2.1 Descripció del mètode

Aquest mètode consisteix en usar un classificador on, en cada frame, s'agafen mostres positives a prop de la posició del quadre delimitador on es troba l'objecte i mostres negatives lluny del centre de l'objecte per actualitzar el classificador. Per predicir la posició de l'objecte en el següent frame, obtenim algunes mostres

a prop del quadre delimitador del frame actual i es determina la que té la puntuació màxima de classificació[1].

A continuació mostrem un resum del procés d'aquest mètode de tracking:

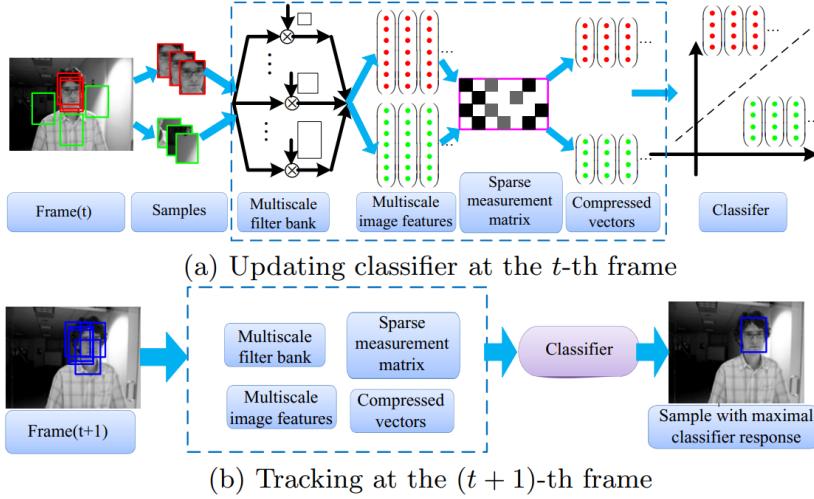


Figura 1: Components principals de l'algorisme de tracking [1]

També mostrem el pseudocodi de l'algorisme per fer-lo més entenedor:

Algorithm 1. Compressive Tracking

Input: t -th video frame

1. Sample a set of image patches, $D^\gamma = \{\mathbf{z} | \|l(\mathbf{z}) - l_{t-1}\| < \gamma\}$ where l_{t-1} is the tracking location at the $(t-1)$ -th frame, and extract the features with low dimensionality.
2. Use classifier H in (4) to each feature vector $\mathbf{v}(\mathbf{z})$ and find the tracking location l_t with the maximal classifier response.
3. Sample two sets of image patches $D^\alpha = \{\mathbf{z} | \|l(\mathbf{z}) - l_t\| < \alpha\}$ and $D^{\zeta, \beta} = \{\mathbf{z} | \zeta < \|l(\mathbf{z}) - l_t\| < \beta\}$ with $\alpha < \zeta < \beta$.
4. Extract the features with these two sets of samples and update the classifier parameters according to (6).

Output: Tracking location l_t and classifier parameters

Figura 2: Pseudocodi de l'algorisme de tracking [1]

2.2.2 Problemes trobats

Comparat amb l'anterior mètode propi, els resultats que obté aquest algorisme són molt millors per la majoria de vídeos que analitzem (veure Annex per fer la comparació entre els dos mètodes de tracking). Però ens ha sobrat que en alguns vídeos el mètode propi obtenia molts millors resultats. Hem investigat

què passa en aquells vídeos quan els executem amb aquest mètode i hem arribat a la conclusió d'un problema que presenta aquesta tècnica:

2.2.2.1 Pèrdua de l'objecte durant el tracking

A l'igual que en el mètode de tracking anterior, aquest mètode també té problemes a l'hora de seguir un objecte quan s'intersecta un altre objecte a prop del quadre delimitador.

Mostrarem un exemple gràfic perquè s'entengui millor. En el vídeo *Alladin*, en un moment determinat estem seguint a la persona però al cap d'uns segons, aquesta persona s'apropa a un objecte i l'algorisme passa a trackejar aquell objecte, perdent de vista a la persona. Això es pot veure en les següents figures.

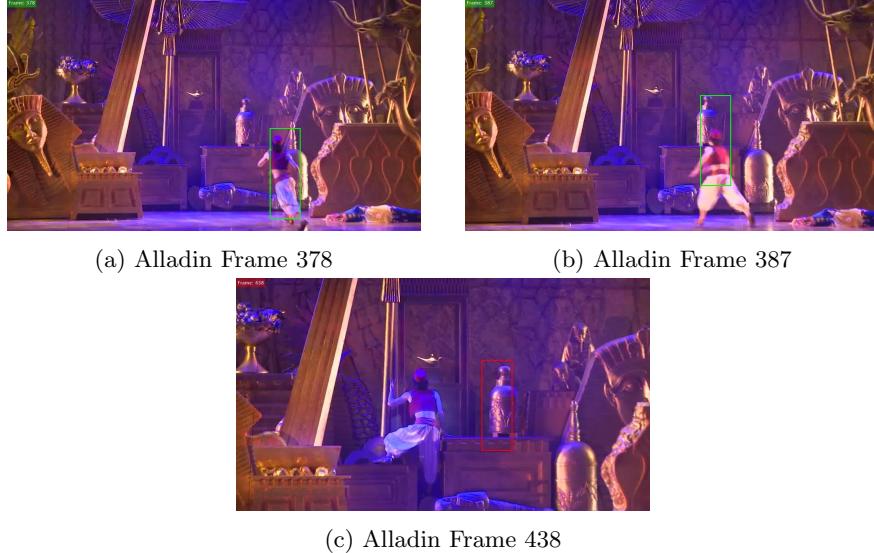


Figura 3: Pèrdua de l'objecte en la seqüència *Alladin*

Aquest error causa que durant molts frames no s'estigui trackejant l'objecte principal. En el nostre cas, però, en el moment en que el quadre delimitador obtingut per tracking i el quadre delimitador original coincideixen en menys d'un 5% decidim parar el vídeo i informar a l'usuari que hi ha hagut una pèrdua total en el frame corresponent. Per la qual cosa el frame 438 no es mostraria en la versió final del codi. Clarament si es vol fer que s'executi tot el vídeo es pot comentar aquesta part del codi i llavors veurem el vídeo fins el final.

2.2.3 Funcions utilitzades

- **classifierUpdate(posx, negx, mu1, sig1, mu0, sig0, lRate)**: Aquesta funció rep com a paràmetres *posx* (conjunt de mostres positives), *negx*

(conjunt de mostres negatives), μ_1 i μ_0 (mitjanes actuals de les característiques positives i negatives), σ_1 i σ_0 (desviació estàndard actual de les característiques positives i negatives) i $lRate$ (taxa de modificació constant).

Actualitza les mitjanes i desviacions estàndard del classificador gaussià utilitzant les mostres positives i negatives, juntament amb els paràmetres actuals. Aquesta actualització permet adaptar el classificador a noves dades i millorar la seva capacitat de classificació en futures iteracions.

- **getFtrVal(iH, samples, ftr):** Aquesta funció rep com a paràmetres iH (imatge integral d'entrada), $samples$ (mostres templates) i ftr (plantilla de característiques).

A partir d'aquests valors calcula les característiques de les mostres utilitzant la imatge integral i els paràmetres de la plantilla de característiques. Retorna una matriu $samplesFtrVal$ amb una mida de $M \times N$, on M és el nombre de característiques i N és el nombre de mostres.

- **HaarFtr(clfparams, ftrparams, M):** Aquesta funció rep com a paràmetres $clfparams$ (paràmetres del classificador), $ftrparams$ (paràmetres de la característica) i M (nombre total de característiques).

Calcula les característiques Haar per al classificador, generant les coordenades, amplades, alçades i pesos corresponents per a un nombre total de característiques determinat. Aquestes característiques seran utilitzades per al processament i la classificació de les imatges en l'algorisme en què s'utilitza aquesta funció.

- **sampleImg(img, initstate, inrad, outrad, maxnum):** Aquesta funció rep com a paràmetres img (imatge d'entrada), $initstate$ (posició de l'objecte), $inrad$ (radi exterior de la regió), $outrad$ (radi interior de la regió) i $maxnum$ (nombre màxim de mostres).

Calcula les coordenades de les plantilles d'imatges de mostres a partir d'una imatge d'entrada, una posició inicial, radi de regió i un nombre màxim de mostres. Aquestes mostres s'utilitzen per al seguiment d'objectes.

3 Reconeixement

En aquesta tasca concreta, s'ens demana desenvolupar un sistema de detecció que analitzi les imatges i extregui un conjunt de característiques per a facilitar la identificació de l'objecte utilitzant tècniques de reconeixement. Encara que disposem d'informació sobre les imatges en el fitxer *groundtruth_rect.txt*, és important destacar que no podem utilitzar aquesta informació durant el procés de reconeixement.

3.1 Mètode Propi

En aquesta subsecció, descriurem el mètode que hem desenvolupat personalment per abordar aquesta tasca específica.

En crear aquest mètode, vam decidir utilitzar principalment els conceptes i les tècniques que s'han ensenyat a classe. L'objectiu era posar a prova el coneixement adquirit i garantir que el mètode fos el més original possible. Tot el codi implementat en aquest mètode és totalment propi, basant-nos exclusivament en els conceptes i les idees presentades durant les sessions de classe.

3.1.1 Descripció del mètode

El mètode segueix un procés detallat que inclou la generació de diverses mostres de l'objecte en qüestió. Aquestes mostres es fan servir per entrenar un classificador, el qual posteriorment s'utilitza per detectar l'objecte en les imatges.

A continuació, es proporcionarà una explicació més detallada del mètode:

3.1.2 Etapes del treball realitzat

Aquest mètode es pot dividir en 3 seccions principals:

1. Entrenament del classificador
2. Prediccions sobre les imatges
3. Càlcul de la precisió

3.1.2.1 Entrenament del classificador

Hem realitzat diverses proves per entrenar el nostre classificador i ara descriuirem el mètode final que hem adoptat, juntament amb altres enfocaments que hem explorat.

Per començar, el procés d'entrenament del classificador consisteix en generar mostres juntament amb les seves etiquetes. Per fer això, prenem cada frame de la imatge i generem un nombre determinat de mostres positives i negatives. Aquest nombre es pot ajustar segons les nostres necessitats. Després de diverses proves, hem trobat que utilitzar 20 mostres positives i negatives per imatge

produceix bons resultats i permet un entrenament eficient. Les mostres positives són aquelles que tenen una superposició del 70% o més amb la finestra correcta, mentre que les mostres negatives tenen una superposició inferior al 30%.

Aquests percentatges són hiperparàmetres que hem ajustat i hem comprovat que aquests valors funcionen bé. A l' hora d'entrenar el model, utilitzem la informació proporcionada per la variable *groundtruth_rect* per obtenir les dimensions de la finestra i generar les mostres, així com per validar la superposició generada.

Un cop hem generat les finestres positives i negatives, apliquem un processament de *feature engineering* bàsic. Redimensionem les finestres a una mida de 64x64 (un altre hiperparàmetre que hem ajustat) per assegurar que les mostres tinguin una mida estàndard. A continuació, apliquem un descriptor a les imatges i normalitzem els resultats obtinguts.

Finalment, després de generar totes les mostres juntament amb les seves etiquetes corresponents, procedim a entrenar el classificador.

Les *Validation Confusion Matrix* de l'entrenament del model per a cada seqüència es poden trobar a l'annex *B.1*.

3.1.2.2 Prediccions sobre les imatges

Aquesta secció ha presentat diversos desafiaments que hem hagut de solucionar i ajustar al llarg del mètode. Detallarem aquests problemes més endavant a la secció *3.1.4*, però també proporcionarem un context aquí.

Per realitzar el reconeixement en les imatges, no podíem conèixer la mida de la finestra amb antelació. Per tant, el primer pas que fem és provar diverses mides de finestra diferents sobre la primera imatge i seleccionar aquella que produueixi els millors resultats pel classificador. Això és possible gràcies al fet que el classificador no només retorna l'etiqueta (0 o 1), sinó també una puntuació *score* que indica el grau de confiança en aquesta predicció. Utilitzem aquesta mida com a mida inicial de la finestra.

Després, per cada frame de la imatge, busquem la finestra utilitzant un hiperparàmetre anomenat *stepSize*, el qual hem ajustat al llarg del procés. Per buscar la finestra, si l'hem trobada al frame anterior, només la busquem en les proximitats d'on l'hem trobada anteriorment, per accelerar el procés, ja que sinó el temps d'execució seria molt elevat. Però, si no hem trobat la finestra al frame anterior, busquem en tota la imatge i, una vegada més, seleccionem la finestra que produueixi la major precisió, evitant així falsos positius abans de trobar l'objecte.

Un cop hem localitzat l'objecte, comprovem si cal ajustar la mida de la finestra. Ho fem analitzant si augmentar la mida de la finestra en un 5% en alguna

de les seves dimensions produeix una millora en el rendiment del classificador. Aquesta tècnica ens permet capturar adequadament situacions en què hi ha un efecte de zoom al vídeo.

3.1.2.3 Càlcul de la precisió

Finalment, un cop tenim les prediccions de la ubicació de l'objecte per a cada fotograma, calculem la precisió del model en aquella seqüència. Utilitzem el següent mètode per fer-ho: calculem la intersecció entre les dues caixes delimitadores i la dividim per l'àrea mínima de les dues caixes. Considerem una predicció encertada quan aquest càlcul ens dóna un valor igual o superior a 0,75.

Hem optat per utilitzar la intersecció entre l'àrea mínima en lloc de la intersecció entre l'àrea total perquè creiem que això reflecteix millor el rendiment dels nostres mètodes. Com que el mètode no és perfecte, sovint la caixa predirà pot tenir una certa "vibració" o ser una mida més petita, però encara segueix correctament l'objecte. Si utilitzéssim la intersecció entre l'àrea total en aquests casos, consideraríem moltes prediccions correctes com a incorrectes.

3.1.3 Estudi del descriptor i el classificador

En aquesta secció es compararan els diferents descriptors i classificadors provats. S'han intentat usar descriptors i classificadors diferents i variats.

Descriptors provats:

- **SIFT**: Descriptor robust dels punts d'interès, que és invariant a canvis d'escala, orientació i distorsió.
- **HOG**: Descriptor robust de les característiques locals d'orientació dels gradients en petites cel·les de la imatge. És invariant a canvis d'orientació i proporciona certa invariància a canvis d'escala i distorsió en les imatges.

Classificadors provats:

- **Decision Tree**: Model predictiu que usa una estructura d'arbre per prendre decisions.
- **Linear Discriminant**: Tècnica estadística per a la classificació de dades basada en l'establiment de fronteres lineals.
- **Naive Bayes**: Mètode de classificació probabilístic basat en el teorema de Bayes amb una suposició de independència entre les característiques.
- **SVM**: Model d'aprenentatge supervisat que usa espais de característiques de gran dimensió per a la classificació.
- **KNN**: Mètode de classificació basada en la distància on els elements són classificats segons la seva semblança amb els K elements més propers en l'espai de característiques.

- **Boosted Tree Ensemble:** Aquesta és una tècnica de modelatge predictiu que combina múltiples arbres de decisió per millorar la precisió de les prediccions. L'enfocament "boosting" busca crear un model fort a partir de la combinació de múltiples models febles, potenciant iterativament la capacitat de classificació del conjunt.
- **Neural Network:** Model computacional inspirat en les xarxes neuronals biològiques, capaç d'aprendre patrons complexos a partir de dades d'entrada.
- **Kernel SVM:** Extensió del SVM que permet tractar dades que no són linealment separables. Mitjançant l'ús d'una funció de kernel, com ara un kernel gaussià, les dades són mapejades a un espai de dimensions més altes on es pot trobar una frontera de decisió lineal.

Igual que al mètode propi de tracking, per l'estudi hem decidit seleccionar seqüències d'imatges de la base de dades de TinyTLP que tinguessin diferents característiques entre elles: en algunes es troben dificultats per detectar l'objecte a causa del nivell d'il·luminació, en altres l'objecte fa moviments bruscs i es pot perdre amb facilitat, i en alguns l'objecte canvia bastant d'aparença dificultant la detecció de característiques importants. Les seqüències d'imatges usades han estat: *MotorcycleChase* (moviments bruscs), *Bike* (canvi d'aparença i moviments bruscs) i *Violinist* (il·luminació).

Per a les diverses combinacions, vam realitzar proves amb aquestes tres seqüències. Vam extreure mostres segons s'ha explicat en la secció anterior i vam entrenar diversos classificadors utilitzant aquestes mostres. En la selecció de la millor combinació, varem tenir en compte el temps necessari per extreure les mostres (ja que és representatiu del temps requerit per a cada imatge), el temps d'entrenament dels classificadors i la seva precisió.

3.1.3.1 Resultats del entrenament

Amb tots els descriptors s'han generat 20 mostres positives i 20 negatives per frame, amb 600 frames, i per tant s'han generat 24000 mostres per seqüència.

1. SIFT

En primer lloc, hem obtingut les mostres mitjançant *SIFT*. Aquests punts d'interès són descrits per un vector de 128 dimensions, que conté informació detallada sobre la seva aparença local.

Dels punts que aconseguim per *SIFT* de cada mostra, hem triat els 10 punts més forts, cadascun amb de 128 dimensions, i el que ens hem guardat és el vector de característiques format per la mitja d'aquests deu punts, en cada dimensió. Això significa que, per a cada mostra, tenim una sola descripció de 128 dimensions, que condensa la informació d'aquests 10 punts d'interès més forts.

Temps per generar les mostres: 566 segons.
 Els resultats dels classificadors es mostren a la següent taula:

Mètode	MotorcycleChase		Bike		Violinist	
	%	Temps (s)	%	Temps (s)	%	Temps (s)
Decision Trees	94	6,7	82,1	13,9	84,7	11,2
Linear Discriminant	92,5	4,7	88,6	7,3	85,5	6
Naive Bayes	86	12	77,51	5,7	82,5	4,5
SMV	98,8	64	96	91,5	92	92
KNN	97,7	134	96,6	56,2	91	51,7
Boosted Tree Ens,	94,6	169	87,5	95,6	90,5	92,2
Neural Network	98,5	653	96,2	137,7	92,6	108,5
Kernel SVM	98,1	455	95	150,9	91,1	138,2

Taula 3: Estudi de descriptors i classificadors, SIFT

I en la següent taula es mostren les mitjanes entre els vídeos, ordenant els classificadors en ordre descendent de precisió:

Mètode	Precisió (%)	Temps (s)
SMV	95,6	82,5
KNN	95,1	80,63
Neural Network	95,76	299,73
Kernel SVM	94,73	248,03
Linear Discriminant	88,86	6
Decision Trees	86,93	10,6
Naive Bayes	82	7,4

Taula 4: Estudi de descriptors i classificadors, mitjana dels vídeos, SIFT

2. HOG

Seguidament, hem obtingut les mostres mitjançant *HOG*. Aquest procediment comença dividint la imatge inicial de 64x64 píxels en cel·les més petites de 8x8 píxels, resultant en un total de 64 cel·les (8x8).

Per a cada una d'aquestes cel·les, es calcula un histograma d'orientacions de gradient, amb un total de 9 bins per histograma, generant un descriptor de característiques per a cada cel·la.

Després, es defineix un bloc com un conjunt de 2x2 cel·les, i es mouen aquest bloc a través de la imatge amb un pas d'1 cel·la a la vegada a causa de l'ajust de 'blockoverlap'. Això dona lloc a un total de 7x7 blocs dins de la imatge.

Per a cada bloc, es calcula un vector de característiques, que és el conjunt de tots els histogrames (normalitzats) de les cel·les dins del bloc. Com que cada histograma conté 9 bins i hi ha 4 cel·les per bloc, això dona un vector de característiques de $4 \times 9 = 36$ dimensions per a cada bloc.

Així, tenim un total de 7×7 blocs * 36 característiques per bloc, la qual cosa resulta en 1764 característiques per a la imatge.

Temps per generar les mostres: 44 segons.

Els resultats dels classificadors es mostren a la següent taula:

Mètode	MotorcycleChase		Bike		Violinist	
	%	Temps (s)	%	Temps (s)	%	Temps (s)
Decision Trees	99,4	39	96,9	120,49	93,3	561
Linear Discriminant	100	104	99,9	365,67	99,5	609
Naive Bayes	99,8	325	93,3	93,43	96,1	262
SVM	100	124	100	304,86	100	630
KNN	100	997	100	2405,30	100	1848
Boosted Tree Ens,	99,4	2314	99,6	4605,6	98,9	4191
Neural Network	100	1965	100	2283,20	99,9	1135
Kernel SVM	100	2400	99,8	8079,5	99,5	8079,5

Taula 5: Estudi de descriptors i classificadors, HOG

I en la següent taula es mostren les mitjanes entre els vídeos, ordenant els classificadors en ordre descendent de precisió:

Mètode	Precisió (%)	Temps (s)
SMV	100	264,7
KNN	100	1312,5
Neural Network	99,97	1345,8
Kernel SVM	99,77	4679,6
Linear Discriminant	99,8	269,6
Boosted Tree Ens,	99,3	2777,6
Naive Bayes	96,4	170,1
Decision Trees	96,53	180,1

Taula 6: Estudi de descriptors i classificadors, mitjana dels vídeos, HOG

Comparació i elecció

Aquesta taula resumeix els dos mètodes:

Generalment, els dos descriptors funcionen bé amb la majoria de models. Usar *SIFT* té un cost temporal significativament menor, i, en canvi, *HOG* dona resultats lleugerament millors. També s'ha de tenir en compte que generar les mostres per *SIFT* ha suposat 566 segons, mentre per *HOG* només 55, el que és 10.29 vegades més ràpid. Això ens indica que clarament serà més ràpid en temps d'execució fer servir *HOG*.

En comparar detalladament tots els models, és clar que el classificador *SVM* és el que té millor equilibri entre resultat i temps, i finalment ens hem decantat

Mètode	SIFT		HOG	
	%	Temps (s)	%	Temps (s)
SMV	95,6	82,5	100	264,7
KNN	95,1	80,6	100	1312,5
Neural Network	95,76	299,7	99,96	1345,8
Kernel SVM	94,73	248	99,76	4679,6
Linear Discriminant	88,86	118,9	99,8	269,6
Boosted Tree Ens,	90,86	6	99,3	2777,6
Naive Bayes	82	10,6	96,4	170,1
Decision Trees	86,93	7,4	96,53	180,12

Taula 7: Estudi de descriptors i classificadors, SIFT i HOG

per fer servir *HOG* com a descriptor, ja que aconsegueix un impressionant 100% d'acerç trigant relativament poc més que *SVM* amb *SIFT*, i a sobretot perquè és molt més ràpid calcular *HOG* en execució.

S'ha de tenir en compte, però, que aquests resultats els obtenim només en *l'entrenament del classificador*. És possible que posteriorment aconseguim pitjors resultats en buscar l'objecte en les seqüències, i es comentaran les raons més endavant.

3.1.4 Problemes trobats

Al dissenyar aquest mètode pràcticament des de zero, ens hem trobat molts problemes, però considerem haver resolt la majoria satisfactòriament. A continuació es llisten els més rellevants, en ordre segons les etapes del mètode:

3.1.4.1 Realitzar *feature engineering*

També vam tenir problemes n'extraurà bé les mostres de les imatges, ja que el classificador a vegades donava coses estranyes perquè les característiques que triàvem no eren prou representatives. Inicialment, només li donàvem les regions que ens indicava el *groundtruth_rect.txt* com a característiques positives, però clarament no funcionava bé.

Després vam provar de generar varíes positives i varíes negatives per frame, i per poder generar varíes positives diferents vam buscar que interaccionin en més d'un 90% amb el correcte. Això ens va portar al problema que era molt lent generar aleatoriament les features, ja que a vegades trigava a triar justament al voltant de l'objecte, i això ho vam solucionar buscant només al voltant de la finestra proporcionada com a correcte, i ara el procés és molt ràpid.

3.1.4.2 Triar una finestra inicial

Com que no tenim accés a res de *groundtruth_rect.txt*, no sabíem amb quina mida de finestra començar. Al principi vam provar una mida arbitrària, però

clarament no funcionava al tenir els objectes a detectar mides tan diferents. Després de moltes voltes vam decidir buscar exhaustivament una bona finestra inicial sobre la primera imatge, i prendre aquesta com a referència, el qual ha funcionat bastant bé i ha solucionat el problema en algúns casos.

3.1.4.3 Zooms en les imatges

Un altre gran problema que hem solucionat en molts casos és que a vegades perdem l'objecte quan hi ha zoom. Per solucionar això el que fem és, una vegada trobem l'objecte, mirem si modificar la mida de la finestra en un 5% en alguna combinació de x i y millora el *score* que dona el classificador, i si ho fa, quedar-nos amb aquesta nova mida. Quan hi ha canvis molt bruscs de zoom, però, no acaba funcionant bé.

3.1.5 Funcions utilitzades

Aquesta és la descripció de les funcions que s'han creat:

- **main**: Funció que executa tot el sistema de reconeixement.
- **calculateAccuracy**: Funció que rep els valors reals, els predictius i el nombre d'elements. Retorna el % d'encert, una llista amb els frames on hem encertat i una llista amb els frames on no hem encertat.
- **visualizeSpecificFrames**: Rep el directori d'imatges, les prediccions, els valors reals i una llista de frames. Permet visualitzar el valor predictiu i el real sobre la llista de frames que li passem.
- **visualizeResults**: Funció que rep el directori d'imatges, les prediccions i els valors reals, i permet visualitzar sobre tot el vídeo el valor predictiu vs. el real per cada frame.
- **makePredictions**: Rep el classificador, el directori d'imatges i un booleà. Retorna les prediccions sobre les imatges fetes pel classificador. Si el booleà és true, es visualitza per cada frame la predició mentre es va fent. Útil per mirar si "va bé" des del principi, ja que predir sobre tota la seqüència és costós en temps.
- **optimizeWindowSize**: Rep la imatge actual, el millor window actual, el seu *score* donat pel classificador, les coordenades on es troba, el window size actual i el classificador. Retorna el window size optimitzat i la millor window. Si no hi ha hagut optimització, aquesta és la mateixa que la que se li ha passat com a paràmetre.
- **findInitialWindow**: Rep el classificador i una imatge, i retorna el millor window size per aquella imatge segons el classificador.

- **trainModel**: Rep els valors reals, el directori d'imatges i un enter. Entrena i retorna un model, pel qual s'han generat tantes mostres com el paràmetre enter per cada frame de la seqüència d'imatges. També retorna totes les mostres amb els seus labels, per si es vol experimentar amb aquestes.
- **genFeatures**: Rep una imatge, una caixa envoltant i un enter. Genera sobre la imatge tantes mostres positives i negatives com indica l'enter, basant-se en la caixa envoltant per determinar el tipus de cada mostra.
- **generateRandomROI**: Rep una imatge, una caixa envoltant i el tipus que es vol generar, el qual és positiu. Retorna una mostra del tipus que s'ha indicat pel paràmetre corresponent.

3.2 Mètode No Propi

En aquesta secció s'explicarà com s'ha realitzat el mètode no propi per a reconeixement.

3.2.1 Descripció del mètode

Aquest mètode consistirà en usar un classificador basat en deep-learning ja entrenat per detecció d'objectes i afinar-lo per a la nostra tasca. Farem servir la llibreria *ultralytics* i el model de classificador *yolov8* preentrenat per detecció d'objectes.[2]

3.2.2 Etapes del treball fet

Aquest mètode, al ser mitjançant una llibreria, ha sigut molt més senzill de fer. Es poden distingir 3 etapes novament, però aquestes han sigut molt més curtes:

1. Preparació de les dades
2. Entrenament del model
3. Càcul de la precisió

3.2.2.1 Preparació de les dades

En el dataset, les dades d'on és l'objecte se'ns donàvem seguint el format *[ID del frame, xmin, ymin, amplada, alçada, isLost]*. D'altra banda, la llibreria espera que només li passem les coordenades del centre de l'objecte, l'amplada i l'alçada, tenint un *.txt* per a cada imatge. Hem fet un preprocessat on generem aquests *.txt's* correctament, i on eliminem aquelles imatges on l'objecte no és visible, per a no confondre al classificador.

3.2.2.2 Entrenament del model

En aquesta part simplement generem l'arxiu *.yaml* per indicar al classificador on són les dades i quines són les classes i després entrenem el model.

Hem decidit afegir unes quantes imatges sobre que fa el model per entrenar, ja que les hem trobat interessants:

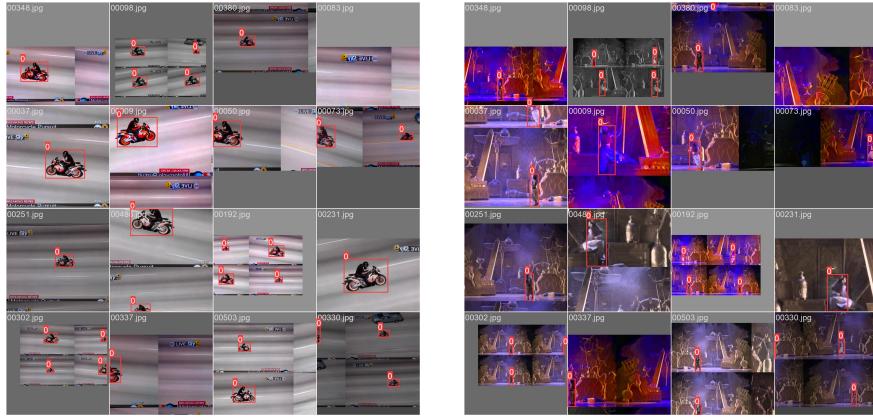


Figura 4: Entrenament dels classificadors

Pel que sembla, la llibreria realitza transformacions a les imatges per ampliar el *dataset* que li passem.

Les *Validation Confusion Matrix* i diversos resultats interessants (que ens dona el model) de l'entrenament del model de cada seqüència es troben en l'annex B.1

3.2.2.3 Càcul de la precisió

Finalment, usant el nostre classificador ja entrenat, recorrem totes les imatges predient on és l'objecte, i agafant la posició real calculem la *Intersection Over Min*. Si aquest és $\geq 0,75$, és una predicció correcta. Calclem la precisió com $\frac{\text{correctes}}{\text{totals}}$, obviament no contant els frames on no es veu l'objecte.

3.2.3 Problemes trobats

Els problemes trobats en aquest mètode han sigut principalment a l'hora d'entendre la llibreria i com se'ns demanaven les coses. No sabíem, per exemple, que la manera que *ultralytics* demanava les coordenades de l'objecte era diferent i ens ha portat algun mal de cap.

Un altre problema que hem tingut és el temps d'entrenament. En ser un classificador molt potent, per molt que fos preentrenat trigava molt, i ho hem solucionat buscant una versió d'aquest que tingués un bon *trade – off* entre

resultats i temps, decantant-nos finalment per la versió *YOLOv8n* de detecció d'objectes.

3.2.4 Funcions utilitzades

S'ha treballat en una *Notebook* de *GoogleCol*, utilitzant les següents llibreries:

- **Ultralytics**: Llibreria de codi obert per al desenvolupament d'algorismes d'aprenentatge profund, especialitzada en visió per computador.
- **Numpy**: Llibreria per a Python que suporta arrays i matrius de gran dimensió, juntament amb una gran col·lecció de funcions matemàtiques per operar amb aquests arrays.
- **cv2**: També coneguda com OpenCV, és una llibreria de processament d'imatges i visió per computadora.
- **glob**: Mòdul en Python que troba tots els camins de fitxer que coinciden amb un patró específic, segons les normes utilitzades pel shell de Unix.
- **os**: Mòdul en Python que proporciona funcions per interactuar amb el sistema operatiu, incloent-hi la lectura, escriptura i canvi de directoris de fitxers.
- **yaml**: Mòdul en Python per treballar amb fitxers YAML, que són útils per a la configuració de dades i la serialització.

I aquesta és la descripció de les funcions que s'han creat:

- **setup**: Aquesta funció rep el nom d'un directori i canvia el directori estàndard a aquest per fer més fàcil gestionar els paths. A més, carrega el model preentrenat i el retorna.
- **preprocess**: Aquesta funció rep el nom de la seqüència que volem tractar i realitza el preprocessat de dades per aquesta, generant els fitxers necessaris per a l'entrada de la llibreria. També s'encarrega d'esborrar les imatges on no es mostra l'objecte.
- **genYAMLandTrain**: Aquesta funció rep el directori (no hem pogut evitar que aquí calgui passar-li), model preentrenat i el nom de la seqüència, i el que fa és generar l'arxiu *.yaml* correcte que necessita el classificador com entrada i entrena aquest.
- **calculateAccuracy**: Aquesta funció rep el nom de la seqüència, el model i un booleà *show*. Prediu on és l'objecte en cada imatge de la seqüència on aquest és visible. Si *show = True*, es mostra per cada imatge la caixa envoltant correcte i la predita.

- **bboxOverlayRatio:** Funció auxiliar que es comporta igual que la que es diu igual a *Matlab*. Rep dues caixes envoltants i el tipo de ratio a calcular, i en funció d'aquest calcula la *Intersection Over Union* o la *Intersection Over Min*. Tot i no usar-se, s'ha afegit la primera opció per completenessa i generalitat de la funció.
- **main:** Funció que executa tot el sistema de reconeixement, i des d'on es toquen els paràmetres.

4 Resultats

4.1 Resultats Tracking Mètode Propi

S'ha utilitzat MATLAB amb Windows 10 per executar el codi i s'ha mesurat el temps d'execució de cada vídeo mitjançant l'eina *Tic-Toc* que ofereix MATLAB.

Per cada vídeo s'ha recollit el percentatge de frames encertats i el temps d'execució. A continuació, es mostren els resultats obtinguts amb aquest mètode en la següent taula:

Vídeo	Precisió (%)	Primera pèrdua	Temps (s)
Alladin	2	61	2,35
Bike	52	-	28,39
Boat	100	-	21,09
CarChase1	100	-	25,55
Dashcam	30	-	19,75
DriftCar2	33	-	22,5
Drone1	53	513	19,83
Drone2	89	-	21,14
Drone3	60	424	14,41
Jet3	100	-	34,06
Mohiniyattam	14	146	5,24
MotorcycleChase	99	-	28,1
PolarBear1	7	63	2,66
Puppies1	3	285	13,79
Violinist	23	-	19,66

Taula 8: Resultats Mètode de Tracking Propi

És important destacar que quan el percentatge de frames encertats és molt baix, el temps d'execució també ho serà. Això és causa de la nostra decisió de suspendre l'algorisme de tracking quan es perd l'objecte, ja que considerem que el tracking ha fallat i no és necessari continuar.

4.1.1 Anàlisi

Per avaluar els resultats, hem realitzat proves amb 15 seqüències d'imatges de dificultat creixent utilitzant el nostre algorisme de tracking. En alguns casos, hem obtingut resultats excel·lents, però en altres casos, els resultats podrien ser millorables. Hem posat tot el nostre esforç per aconseguir resultats acceptables en tots els vídeos, però amb els recursos que teníem disponibles vists en l'assignatura, no disposàvem de més eines que poguessin complementar el nostre algorisme.

Destaquem el rendiment excepcional en les seqüències d'imatges *Boat*, *CarChase1*, *Jet3* i *MotorcycleChase*, ja que hem pogut fer el seguiment de l'objecte

de manera contínua durant tota la seqüència sense perdre'l. Després d'un estudi sobre aquestes seqüències, hem observat que els objectes en qüestió es distingeixen clarament del fons en què es troben. Això ens porta a concloure que el nostre algorisme té un rendiment excel·lent en aquestes situacions.

A continuació, mostrem alguns exemples de les seqüències *Boat* i *Jet3*, on es pot apreciar com l'objecte es distingeix clarament del fons:



Figura 5: Excel·lent rendiment amb objecte distingut del fons

En alguns vídeos, observem un rendiment mitjà, ja que obtenim un baix percentatge d'encerts, però som capaços de mantenir el seguiment de l'objecte durant tota la seqüència. Després d'un estudi exhaustiu, hem identificat la causa d'aquest baix percentatge d'encerts. Resulta que, tot i que seguim adequadament l'objecte durant tota la seqüència, en algun moment el quadre delimitador de l'objecte es desplaça en una direcció i el centre de l'objecte s'allunya del centre del quadre delimitador.

Aquest problema es pot apreciar en l'exemple extret de la seqüència *Bike*, on podem observar com passem de fer el seguiment de la persona a només fer el seguiment de la roda posterior de la bicicleta.

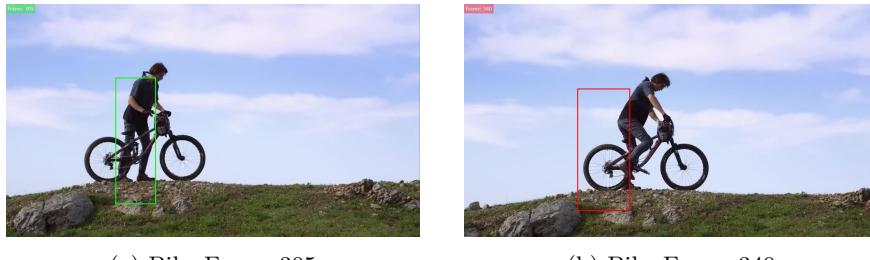


Figura 6: Rendiment mitjà en la seqüència *Bike*

Finalment, hi ha seqüències d'imatges en les quals obtenim uns resultats molt pobres. Prenem com a exemple la seqüència *Mohiniyattam*: en un dels frames inicials, aconseguim detectar amb precisió la persona, però gradualment

comencem a desplaçar la caixa delimitadora cap a una altra direcció. En els frames següents, encara som capaços de detectar la persona, però quan aquesta realitza un moviment brusc cap a la dreta, ja no som capaços de detectar-la.

Aquesta situació posa de manifest les limitacions del nostre algorisme en casos de moviments ràpids o canvis bruscs de direcció quan ja tenim el quadre delimitador desplaçat del centre de l'objecte. Hem reconegut aquesta situació com a una àrea de millora per al nostre algorisme de seguiment.



Figura 7: Pèrdua de l'objecte de seguiment en *Mohiniyattam*

Encara així, estem bastant contents dels resultats obtinguts, ja que el codi s'ha fet enterament per nosaltres i hi ha vídeos on obtenim molts bons resultats, alguns fins i tot del 100%.

En mitjana el mètode dona un 51% de precisió i triga 24,47 segons en fer tot el procés, si tenim només en compte els temps dels vídeos on hem seguit l'objecte fins al final.

4.2 Resultats Tracking Mètode No Propi

Com en el cas anterior, hem executat el codi MATLAB amb Windows 10 i hem mesurat el temps d'execució amb *Tic-Toc*.

Per cada vídeo mostrem el percentatge de frames encertats i el temps d'execució. A continuació, es mostren els resultats obtinguts amb aquest mètode en la següent taula:

Vídeo	Precisió (%)	Primera pèrdua	Temps (s)
Alladin	65	402	9,52
Bike	29	234	6,53
Boat	100	-	16,81
CarChase1	100	-	16,77
Dashcam	100	-	14,61
DriftCar2	53	407	11,47
Drone1	100	-	17,39
Drone2	100	-	16,89
Drone3	59	375	10,84
Jet3	59	426	12,55
Mohiniyattam	66	-	17,12
MotorcycleChase	100	-	13,56
PolarBear1	20	181	4,85
Puppies1	100	-	16,64
Violinist	96	-	16,55

Taula 9: Resultats Mètode de Tracking Propi

4.2.1 Anàlisi

Observem com ara molts vídeos obtenen un alt percentatge d'encerts i en gairebé tots ells arribem fins al final de la seqüència o, en el pitjor dels casos, a almenys una tercera part.

En els exemples de *Boat*, *CarChase1*, *Jet3* i *MotorcycleChase*, aquest mètode també obté resultats excel·lents.

Per a les seqüències d'imatges que van tenir un rendiment mitjà en el mètode anterior, en alguns casos s'ha pogut solucionar i s'ha obtingut un rendiment espectacular, com és el cas de *Dashcam*, que ara té una precisió del 100%. En altres casos, s'ha millorat una mica la precisió, com és el cas de *DriftCar2*. No obstant això, hi ha un cas en el qual el rendiment ha empitjorat, concretament en la seqüència *Bike*.

Realitzarem un estudi detallat de la seqüència *Bike* per comprendre en què falla aquest mètode.



Figura 8: Dificultat en trackejar *Bike*

Observem com, a diferència del mètode anterior, aquest nou mètode no és capaç de detectar de manera correcta la persona objectiu i, en un moment determinat, comença a fer el seguiment del terra, perdent definitivament l'objectiu original del trackeig. A causa d'això, aquest mètode obté un rendiment més baix en comparació amb el mètode propi que havíem utilitzat anteriorment.

No obstant això, encara hi ha vídeos en els quals és difícil fer el seguiment de l'objecte. La seqüència *PolarBear1*, per exemple, continua obtenint un percentatge d'encerts inferior al 25%. Si analitzem les imatges, podem observar que es tracta d'un os polar sobre un fons blanc, la qual cosa dificulta el seguiment ja que l'objecte i el fons s'assemblen molt.



Figura 9: Dificultat en trackejar *PolarBear1*

En mitjana, aquest mètode proporciona una precisió del 76,47% i triga 13,47 segons en completar tot el procés. És destacable que hem augmentat la precisió en un 25% mentre hem reduït el temps d'execució en 10 segons.

En resum, estem satisfets amb aquest mètode de seguiment no propi, ja que hem aconseguit millorar el rendiment en la majoria de les seqüències en comparació amb el mètode propi, al mateix temps que obtenim temps d'execució més baixos que el mètode propi.

4.3 Resultats Reconeixement Mètode Propi

S'ha utilitzat MATLAB amb Windows 10 per executar el codi i s'ha mesurat el temps d'execució de cada vídeo mitjançant l'eina *Tic-Toc* que ofereix MATLAB.

A continuació, es mostren els resultats obtinguts amb aquest mètode en la següent taula:

Vídeo	Precisió (%)	Temps (s)
Alladin	6	353,9
Bike	5	375,2
Boat	93,5	398,5
CarChase1	97,67	412,3
Dashcam	63,5	380,6
Driftcar2	20,5	395,7
Drone1	1,5	360,9
Drone2	26,67	226,46
Drone3	12,17	390,1
Jet3	88	387,2
Mohiniyattam	99	409,6
MotorcycleChase	98	262,9
PolarBear1	43	403,7
Puppies1	5	379,4
Violinist	91,67	441,4

Taula 10: Resultats Reconeixement Mètode Propi

4.3.1 Anàlisi

Com es pot observar, el mètode funciona molt bé en alguns casos senzills, com ara *Boat* i *MotorcycleChase*, i fins i tot en situacions més complexes, com *Violinist* i *Mohiniyattam*. No obstant això, en analitzar els vídeos on presenta dificultats, hem detectat els següents casos en els quals falla:

1. Quan el zoom canvia de manera molt significativa, com en *Bike*. Això és perquè la nostra manera d'actualitzar el window size només ens permet fer-ho en un 5%, i si el zoom augmenta molt de pressa seguim buscant amb una finestra incorrecta i mai tornem a trobar l'objecte
2. Quan el fons es diferencia poc de l'objecte o és molt ple, com en *PolarBear1* o *Puppies1*. Això succeeix ja que usem *HOG* com a descriptor, i si no podem diferenciar bé els gradients de l'objecte del fons, no el podem trobar correctament.
3. Quan l'objecte és molt petit, com en *Drone3*. La window size mínima és massa gran i mai acaben de trobar bé l'objecte

4. Quan hi ha una cosa amb gradient semblant a l'objecte, com en *Drone1*. El que succeeix en aquest vídeo és graciós, i és que el vídeo segueix a l'ombra de la persona. Això passa perquè per molt que mirem finestres de diferents mides, sempre redimensionem a 64x64, per tant, amb la mida correcta el *HOG* de l'ombra és molt similar al de la persona, per molt que un sigui estirat de manera vertical i l'altre de manera horitzontal, ja que això es perd en fer-ho quadrat.
5. Quan l'objecte es mou molt de pressa també falla a vegades, això es deu al fet que si hem trobat l'objecte al frame anterior, el busquem al voltant d'on l'hem trobat al següent frame. Però si ja no hi és allà per haver-se mogut molt ràpid el perdem, i després ens costa tornar a trobar-lo.

Ens ha sorprès que els resultats siguin tan diferents del que donava a esperar la *ConfusionMatrix* de l'entrenament del classificador. Pensem que això és perquè el classificador *sí* reconeix bé l'objecte, però en condicions molt bones, és a dir, si encertem amb la mida de la finestra i el *StepSize*.

També creiem que es prediuen coses que no som l'objecte com si ho fossin, però coses llunyanes d'aquest, i com en l'entrenament només li donem mostres negatives al voltant de l'objecte (amb l'esperança de tenir una caixa envoltant estable) aquests sí que els confon.

En mitjana el mètode dona un 47,11% de precisió i triga 377'9 segons en fer tot el procés, una mica més de 6 minuts.

4.4 Resultats Reconeixement Mètode No Propi

Hem mesurat el temps mitjançant la llibreria de Python *time*, i hem executat el codi en una *Notebook* de *Google Colab*, usant una *GPU T4*.

Vídeo	Precisió (%)	Temps (s)
Alladin	93,33	334,9
Bike	98,67	321,5
Boat	100	350,7
CarChase1	98,83	368,1
Dashcam	100	238,8
Driftcar2	94,33	348,7
Drone1	100	300,1
Drone2	90,33	332,7
Drone3	61,83	380,3
Jet3	100	295,2
Mohiniyattam	94,5	343,7
MotorcycleChase	100	327,6
PolarBear1	99,83	359,1
Puppies1	100	345,4
Violinist	98,5	334,2

Taula 11: Resultats Reconeixement Mètode No Propi

4.4.1 Anàlisi

No hi ha gaire que comentar sobre aquest mètode, ja que funciona molt bé en tots els vídeos que hem provat. Era d'esperar d'una llibreria tan madura, i un model tan ben entrenat i estudiat com és *YoloV8*, però, tot i això, ens ha sorprès que anés tan bé.

En mitjana el mètode dona un impressionant 95.34% de precisió i triga 332 segons, al voltant de 5 minuts i mig.

5 Conclusió

En general, estem d'acord que aquest treball ens ha estat molt útil per aprendre com funciona el tracking i la detecció d'objectes en vídeos, i quines tècniques són més eficients en diferents escenaris.

Hem pogut observar que, en el nostre cas, si volem obtenir resultats ràpids per al tracking en temps real, és més convenient utilitzar un mètode de tracking. Com hem vist a l'exemple del mètode de tracking no propi, aquesta opció ofereix resultats molt satisfactoris en un temps d'execució breu, amb una mitjana de precisió del 76,47% en només 13,47 segons de promig, el que és impressionant.

D'altra banda, si volem obtenir un resultat robust i evitar errors, hem comprovat que utilitzar un mètode de reconeixement és més adequat. En el cas del mètode de reconeixement no propi, obtenim una mitjana de precisió del 95,34%, tot i que hem de pagar un preu elevat en temps d'execució, amb un promig de 5 minuts i mig.

En conclusió, hem après que seleccionar la tècnica de seguiment més adequada depèn de les nostres necessitats específiques: si prioritzen la velocitat i un rendiment acceptable, el tracking és una opció viable; mentre que si busquen un resultat altament precís i robust, el reconeixement és la millor opció, tot i el cost més elevat en temps d'execució.

Referències

- [1] Real-time Compressive Tracking — www4.comp.polyu.edu.hk. <http://www4.comp.polyu.edu.hk/~cslzhang/CT/CT.htm>. [Accessed 22-Jun-2023].
- [2] Ultralytics — Revolutionizing the World of Vision AI — ultralytics.com. <https://ultralytics.com>. [Accessed 22-Jun-2023].
- [3] Abhinav Moudgil and Vineet Gandhi. Long-term visual object tracking benchmark. In *Asian Conference on Computer Vision*, pages 629–645. Springer, 2018.

A Annex A: Codi

A.1 Codi Tracking Mètode Propi

```
main()

function main()
    % Tanquem tot
    close all; clear all;

    % Selecciona el vídeo que vols executar
    % -----
    dataset = 'MotorcycleChase';
    % ----- %

    % Llegim groundtruth_rect.txt i la seqüència d'imatges
    BB = importdata(['./' dataset '/groundtruth_rect.txt']);
    Idir = dir(['./' dataset '/img/*.jpg']);

    % Prenem el primer frame com a imatge model
    frame = imread(horzcat(Idir(1).folder, '/', Idir(1).name));

    % Obtenim les característiques de la imatge model
    im_obj = rgb2gray(imcrop(frame, BB(1, 2:5)));
    kp_obj = detectSIFTFeatures(im_obj);
    kp_obj = selectStrongest(kp_obj, 5);
    [feat_obj, kp_obj] = extractFeatures(im_obj, kp_obj);

    % Inicialitzem les dades del primer frame
    rect = BB(1, 2:5);
    prev_rect = rect;
    prev_frame = frame;

    % Filtre de Kalman per predir la mida del rectangle
    initialStateW = [BB(1,4) 0 BB(1,5) 0];
    KFW = trackingKF('MotionModel', '2D Constant Velocity', 'State', initialStateW);

    % Creem un directori per guardar les imatges detectades per tracking
    folder_path = ['results_' dataset '_Tracking_Propi'];
    if exist(folder_path, 'dir')
        rmdir(folder_path, 's'); % Eliminem el directori i els seus continguts
    end
    mkdir(folder_path);

    % Declarem variables que ens serviran per l'execució
    overlaps = 1; % Comptador d'overlaps
```

```

% Booleà que indica si hem percut completament l'objecte
total_loss = 0;
tic; % Prenem mesura del temps d'execució

nf = size(Idir);
for i = 2:nf
    % Obtenim la imatge del frame actual
    filename = horzcat(Idir(i).folder, '/', Idir(i).name);
    frame = imread(filename);

    % Reduïm la mida del frame actual a les proximitats de l'anterior rectangle
    rectPerc = getRectanglePercentage(rect, 0.35);
    im_scene = rgb2gray(imcrop(frame, rectPerc));

    % Detecció de punts
    kp_scene = detectSIFTFeatures(im_scene);

    % Seleccionem els millors keypoints
    kp_scene = selectStrongest(kp_scene, 10); % /kp_scene/ >> /kp_obj/

    % Obtenim les característiques
    [feat_sce, kp_scene] = extractFeatures(im_scene, kp_scene);

    % Matching de keypoints
    pairs = matchFeatures(feat_obj, feat_sce, 'MatchThreshold', 35);

    if isempty(pairs)
        % En aquesta part no hem pogut trobar l'objecte per tracking.
        % Tant la tècnica de la predicció de la posició amb el
        % filtre de Kalman, i la tècnica de buscar l'objecte per tota
        % la imatge funcionen pitjor que deixar el rectangle en la
        % posició del frame anterior, així que ho hem deixat així.
    else
        % Seleccionem els keypoints que han fet matching
        m_kp_obj = kp_obj(pairs(:, 1), :);
        m_kp_scene = kp_scene(pairs(:, 2), :);

        % Predictor de l'amplada/alçada de la finestra per Kalman
        predW = predict(KFW);
        correct(KFW, BB(i, 4:5));

        % Obtenim el rectangle dels keypoints
        rect_kp = rectangleFromKP(m_kp_scene.Location, rectPerc);

        % Redimensionem el rectangle dels keypoints amb la mida del predict
        rect = resizeRectangle(rect_kp, predW(1), predW(3));

```

```

% Corregim el rectangle si s'ha mogut lluny de l'anterior
rect = correctRectanglePos(prev_rect, rect);
end

% Calculem l'overlap del rectangle obtingut per tracking
% i la bounding box real de groundtruth_rect.txt
overlap_ratio = bboxOverlapRatio(rect, BB(i, 2:5), 'Min');

if overlap_ratio >= 0.60
    overlaps = overlaps + 1;
    color = 'green';
else
    if overlap_ratio <= 0.05 && BB(i, 6) == 0
        msg_loss = ['Pèrdua total en el frame ' num2str(i) '.\n'];
        total_loss = 1;
        break;
    else
        color = 'red';
    end
end

% Visualization
text_str = ['Frame: ' num2str(i)];
result = insertText(frame, [1 1], text_str, 'FontSize', 15, ...
    'BoxColor', color, 'BoxOpacity', 0.4, 'TextColor', 'white');
result = insertShape(result, 'Rectangle', rect, 'LineWidth', 3, 'Color', color);
imwrite(result, [folder_path num2str(i, '%04i.jpg')]);

prev_frame = frame;
prev_rect = rect;
end

time = toc; % Guardem el temps d'execució

% Informe de l'execució
fprintf('Overlap ratio: %.2f\n', overlaps/nf(1));
if total_loss
    fprintf(msg_loss);
else
    fprintf("No s'ha perdut l'objecte en tot el vídeo\n");
end
fprintf('Temps: %.2f segons\n', time);

% Guardem els resultats en un arxiu de text
info = fopen([folder_path '/_info.txt'], 'w');

```

```

if info == -1
    error('Unable to open file for writing.');
else
    fprintf(info, 'Overlap ratio: %.2f\n', overlaps/nf(1));
    if total_loss
        fprintf(info, msg_loss);
    else
        fprintf(info, "No s'ha percut l'objecte en tot el vídeo\n");
    end
    fprintf(info, 'Temps: %.2f segons\n', time);
    fclose(info);
end
end

function perc_rect = getRectanglePercentage(rect, perc)
% Calcular l'augment en la grandària del rectangle
extraWidth = rect(3) * perc;
extraHeight = rect(4) * perc;

% Calcular les noves coordenades del rectangle
newX = rect(1) - extraWidth / 2;
newY = rect(2) - extraHeight / 2;
newWidth = rect(3) + extraWidth;
newHeight = rect(4) + extraHeight;

% Realitzar el crop utilitzant les noves coordenades
perc_rect = [newX, newY, newWidth - 1, newHeight - 1];
end

function new_rect = resizeRectangle(rect, width, height)
% Extreure les propietats inicials del rectangle
initialWidth = rect(3);
initialHeight = rect(4);
initialCenterX = rect(1) + (initialWidth / 2);
initialCenterY = rect(2) + (initialHeight / 2);

% Calcular el desplaçament en amplada i alçada
deltaWidth = width - initialWidth;
deltaHeight = height - initialHeight;

% Calcular les semidimensions
halfWidth = initialWidth / 2;
halfHeight = initialHeight / 2;

% Calcular les coordenades noves de les cantonades del rectangle
xMin = initialCenterX - halfWidth - (deltaWidth / 2);

```

```

yMin = initialCenterY - halfHeight - (deltaHeight / 2);

% Creem el rectangle nou amb l'amplada i l'alçada desitjades
new_rect = [xMin, yMin, width, height];
end

function rect_kp = rectangleFromKP(kp_coordinates, rect)
    % Calcular els límits del rectangle que engloba els keypoints
    x_min = min(kp_coordinates(:, 1));
    x_max = max(kp_coordinates(:, 1));
    y_min = min(kp_coordinates(:, 2));
    y_max = max(kp_coordinates(:, 2));

    % Calcular l'ample i l'altura del rectangle que engloba els keypoints
    kp_width = x_max - x_min;
    kp_height = y_max - y_min;

    rect_kp = [x_min+rect(1), y_min+rect(2), kp_width, kp_height];
end

function curr_rect = correctRectanglePos(prev_rect, curr_rect)
    % Calcular els centres dels rectangles
    prevCenter = [prev_rect(1) + prev_rect(3)/2, prev_rect(2) + prev_rect(4)/2];
    currCenter = [curr_rect(1) + curr_rect(3)/2, curr_rect(2) + curr_rect(4)/2];

    % Calcular la distància entre els centres
    dist = sqrt(sum((prevCenter - currCenter).^2));

    threshDist = 0.10 * sqrt(sum(prev_rect(3:4).^2)); % Establert al 5% de la longitud diagonal

    % Si la distància és més gran que un llindar
    if dist > threshDist
        % Calcular la direcció des del centre actual fins a l'anterior
        dir = (prevCenter - currCenter) / dist;

        % Calcular el nou centre
        newCenter = currCenter + dir * (dist - threshDist);

        % Ajustar al quadre delimitador actual
        curr_rect(1) = newCenter(1) - curr_rect(3)/2;
        curr_rect(2) = newCenter(2) - curr_rect(4)/2;
    end
end

```

A.2 Codi Tracking Mètode No Propi

```
main()

function main()
    clc; clear all; close all;

    % Selecciona el vídeo que vols executar
    % -----
    dataset = 'MotorcycleChase';
    %
    rand('state',0);
    %
    BB = importdata(['.' dataset '/groundtruth_rect.txt']);
    img_dir = dir(['.' dataset '/img/*.jpg']);
    initstate = BB(1, 2:5);
    %
    img = imread(horzcat(img_dir(1).folder, '/', img_dir(1).name));
    img = double(img(:,:,1));
    %
    trparams.init_negnumtrain = 50; %number of trained negative samples
    trparams.init_postrainrad = 4.0; %radical scope of positive samples
    trparams.initstate = initstate; % object position [x y width height]
    trparams.srchwinsz = 20; % size of search window
    %
    % classifier parameters
    clfparams.width = trparams.initstate(3);
    clfparams.height= trparams.initstate(4);
    %
    % feature parameters
    % number of rectangle from 2 to 4.
    ftrparams.minNumRect = 2;
    ftrparams.maxNumRect = 4;
    %
    M = 50; % number of all weaker classifiers, i.e., feature pool
    %
    posx.mu = zeros(M,1); % mean of positive features
    negx.mu = zeros(M,1);
    posx.sig = ones(M,1); % variance of positive features
    negx.sig = ones(M,1);
    %
    % Learning rate parameter
    lRate = 0.85;
    %
    % Compute feature template
    [ftr.px, ftr.py, ftr.pw, ftr.ph, ftr.pwt] = HaarFtr(clfparams, ftrparams,M);
    %-----
```

```

% Compute sample templates
posx.sampleImage = sampleImg(img, initstate, trparams.init_postrainrad, 0, 100000);
negx.sampleImage = sampleImg(img, initstate, 1.5*trparams.srchwinsz, ...
4+trparams.init_postrainrad, 50);
%-----
%-----Feature extraction
iH = integral(img); %Compute integral image
posx.feature = getFtrVal(iH, posx.sampleImage, ftr);
negx.feature = getFtrVal(iH, negx.sampleImage, ftr);
%-----
% update distribution parameters
[posx.mu, posx.sig, negx.mu, negx.sig] = classifierUpdate(posx, negx, posx.mu, ...
posx.sig, negx.mu, negx.sig, lRate);
%-----
num = length(img_dir); % number of frames
%-----
x = initstate(1); % x axis at the Top left corner
y = initstate(2);
w = initstate(3); % width of the rectangle
h = initstate(4); % height of the rectangle
%-----
% Creem un directori per guardar les imatges detectades per tracking
folder_path = ['results_' dataset '_Tracking_No_Propi'];
if exist(folder_path, 'dir')
    rmdir(folder_path, 's'); % Eliminem el directori i els seus continguts
end
mkdir(folder_path);

overlaps = 1;
total_loss = 0;
tic;
for i = 1:num
    filename = horzcat(img_dir(i).folder, '/', img_dir(i).name);
    img = imread(filename);
    imgSr = img; % imgSr is used for showing tracking results.
    img = double(img(:,:,1));
    detectx.sampleImage = sampleImg(img, initstate, trparams.srchwinsz, 0, 100000);
    iH = integral(img); %Compute integral image
    detectx.feature = getFtrVal(iH,detectx.sampleImage,ftr);
    %-----
    r = ratioClassifier(posx, negx, detectx); % Compute the classifier for all samples
    clf = sum(r); % Linearly combine the ratio classifiers in r to the final classifier
    %-----
    [c,index] = max(clf);
    %-----
    x = detectx.sampleImage.sx(index);

```

```

y = detectx.sampleImage.sy(index);
w = detectx.sampleImage.sw(index);
h = detectx.sampleImage.sh(index);
initstate = [x y w h];

% Actualitzem overlaps
overlap_ratio = bboxOverlapRatio(initstate, BB(i, 2:5), 'Min');

if overlap_ratio >= 0.60
    overlaps = overlaps + 1;
    color = 'green';
else
    if overlap_ratio <= 0.05 && BB(i, 6) == 0
        msg_loss = ['Pèrdua total en el frame ' num2str(i) '.\n'];
        total_loss = 1;
        %break;
        color = 'red';
    else
        color = 'red';
    end
end

% Visualization
text_str = ['Frame: ' num2str(i)];
result = insertText(uint8(imgSr), [1 1], text_str, 'FontSize', 15, ...
    'BoxColor', color, 'BoxOpacity', 0.4, 'TextColor', 'white');
result = insertShape(result, 'Rectangle', initstate, 'LineWidth', 3, 'Color', color);
imwrite(result, [folder_path num2str(i, '%04i.jpg')]);
%-----Extract samples
posx.sampleImage = sampleImg(img, initstate, trparams.init_postrainrad, 0, 100000);
negx.sampleImage = sampleImg(img, initstate, 1.5*trparams.srchwinsz, ...
    4+trparams.init_postrainrad, trparams.init_negnumtrain);
%-----Update all the features
posx.feature = getFtrVal(iH, posx.sampleImage, ftr);
negx.feature = getFtrVal(iH, negx.sampleImage, ftr);
%-----
% update distribution parameters
[posx.mu, posx.sig, negx.mu, negx.sig] = classifierUpdate(posx, negx, posx.mu, ...
    posx.sig, negx.mu, negx.sig, lRate);
end
time = toc; % Guardem el temps d'execució

% Informe de l'execució
fprintf('Overlap ratio: %.2f\n', overlaps/num);
if total_loss
    fprintf(msg_loss);

```

```
else
    fprintf("No s'ha percut l'objecte en tot el vídeo\n");
end
fprintf('Temps: %.2f segons\n', time);
end
```

A.3 Codi Reconeixement Mètode Propi

```

main()

function main()
    % Modificar aquest valor
    sequence = "MotorcycleChase";

    % Read the annotations file groundtruth_rect.txt: frame, bounding boxes, is_lost
    BB = importdata('./' + sequence + '/groundtruth_rect.txt');
    Idir = dir('./' + sequence + '/img/*.jpg');

    % Also returns features and labels too, to test with Matlab's
    % Classification Learner
    [classifier, features, labels] = trainModel(BB, Idir, 20);

    predictions = makePredictions(classifier, Idir, false);

    visualizeResults(Idir, predictions, BB)

    % We also get the correct and incorrect frames in case we want to
    % visualize what went wrong or when it works.
    [accuracy, correct_frames, incorrect_frames] = calculateAccuracy(BB, predictions, numel(Idir));

    % Print accuracy as a percentage with two decimal places
    fprintf('The accuracy is %.2f%%\n', accuracy*100);

    visualizeSpecificFrames(Idir, predictions, BB, incorrect_frames)
end

function [accuracy, correct_frames, incorrect_frames] = calculateAccuracy(BB, predictions, nf)

    % We consider correct when intersection over min >= 75%
    correct = 0;
    incorrect = 0;
    correct_frames = [];
    incorrect_frames = [];
    for i = 1:nf
        % Only process if isLost is 0
        if BB(i,6) == 0
            if isempty(predictions{i})
                incorrect = incorrect + 1;
            else
                bbox_true = BB(i, 2:5);
                bbox_prediction = predictions{i};

```

```

        if bboxOverlapRatio(bbox_prediction, bbox_true, 'Min') >= 0.75
            correct = correct + 1;
            correct_frames = [correct_frames, i];
        else
            incorrect = incorrect + 1;
            incorrect_frames = [incorrect_frames, i];
        end
    end
end
accuracy = correct / (correct + incorrect);
end

function visualizeSpecificFrames(Idir, predictions, true_values, specific_frames)
    close all
    for i = specific_frames
        filename = horzcat(Idir(i).folder,'/',Idir(i).name);
        I = imread(filename);

        imshow(I); hold on; % show the image and hold the figure for drawing

        rectangle('Position',predictions{i}, 'EdgeColor','r');
        rectangle('Position',true_values(i, 2:5), 'EdgeColor','b')
        hold off;
        pause(0.1);
    end
end

function visualizeResults(Idir, predictions, true_values)
    close all
    nf = numel(Idir); % total number of image files

    for i = 1:nf
        filename = horzcat(Idir(i).folder,'/',Idir(i).name);
        I = imread(filename);

        imshow(I); hold on; % show the image and hold the figure for drawing

        rectangle('Position',predictions{i}, 'EdgeColor','r');
        rectangle('Position',true_values(i, 2:5), 'EdgeColor','b')
        hold off;
        pause(0.1);
    end
end

```

```

function predictions = makePredictions(classifier, Idir, visualize)
    close all
    nf = numel(Idir); % total number of image files
    predictions = cell(nf, 1); % Initialize cell array to hold predictions

    firstImage = imread(horzcat(Idir(1).folder, '/', Idir(1).name));
    windowSize = findInitialWindow(classifier, firstImage);

    % Step size for the sliding window
    stepSize = [30, 30];

    object_found_in_previous_frame = false;

    for i = 1:nf
        filename = horzcat(Idir(i).folder, '/', Idir(i).name);
        I = imread(filename);

        object_found_in_current_frame = false;

        if visualize
            imshow(I); hold on; % show the image and hold the figure for drawing
        end

        bestScore = -Inf;

        % Si hem trobat l'objecte al frame anterior, només el busquem al
        % voltant ara.
        if object_found_in_previous_frame

            for x = max(1, bestx>windowSize(1)) : stepSize(1) : ...
                min(size(I, 1)-windowSize(1), bestx + windowSize(1))

                for y = max(1, besty>windowSize(2)) : stepSize(2) : ...
                    min(size(I, 2)-windowSize(2), besty + windowSize(2))

                        % Extract window from the image
                        window = I(x:x+windowSize(1)-1, y:y+windowSize(2)-1, :);
                        window = imresize(window, [64, 64]); % resize

                        % Compute HoG features for the window
                        hogFeatures = extractHOGFeatures(window);
                        % Z-score normalization
                        hogFeatures = (hogFeatures - mean(hogFeatures))/std(hogFeatures);

                        % Classify the window

```

```

[label, score] = predict(classifier, hogFeatures);

% If it's a positive label and the score is higher than current best,
% store this window
if label == 1 && score(2) > bestScore
    bestScore = score(2);
    bestWindow = [y, x, windowHeight(2), windowHeight(1)]; % [x, y, width, height]
    besty = y;
    bestx = x;
    object_found_in_current_frame = true;
end
end
end

else
for x = 1:stepSize(1):size(I,1) - windowHeight(1)
    for y = 1:stepSize(2):size(I,2) - windowHeight(2)
        % Extract window from the image
        window = I(x:x+windowHeight(1)-1, y:y+windowHeight(2)-1, :);
        window = imresize(window, [64, 64]);

        % Compute HoG features for the window
        hogFeatures = extractHOGFeatures(window);
        % Z-score normalization
        hogFeatures = (hogFeatures - mean(hogFeatures))/std(hogFeatures);

        % Classify the window
        [label, score] = predict(classifier, hogFeatures);

        % If it's a positive label and the score is higher than current best,
        % store this window
        if label == 1 && score(2) > bestScore
            bestScore = score(2);
            bestWindow = [y, x, windowHeight(2), windowHeight(1)]; % [x, y, width, height]
            besty = y;
            bestx = x;
            object_found_in_current_frame = true;
        end
    end
end
end

object_found_in_previous_frame = object_found_in_current_frame;

% In this function we see if slightly increasing or decreasing the
% window size makes the prediction better

```

```

[bestWindow, windowHeight] = optimizeWindowSize(I, bestWindow, ...
    bestScore, besty, bestx, windowHeight, classifier);

% Store the best window found
predictions{1} = bestWindow;

if visualize && ~isempty(bestWindow)
    rectangle('Position',bestWindow,'EdgeColor','r');
    hold off;
    pause(0.1);
end
end
end

function [bestWindow, bestWindowSize] = optimizeWindowSize(I, bestWindow, ...
    bestScore, y, x, windowHeight, classifier)

bestWindowSize = windowHeight; % A default value

scaleFactors = [0.95, 1.05]; % Decrease and increase the window size by 10%

for scaleFactorX = scaleFactors
    for scaleFactorY = scaleFactors
        newWindowSize = [round(windowSize(1)*scaleFactorX), round(windowSize(2)*scaleFactorY)];

        endX = min(x+newWindowSize(1)-1, size(I,1));
        endY = min(y+newWindowSize(2)-1, size(I,2));

        newWindow = I(x:endX, y:endY, :);
        newWindow = imresize(newWindow, [64, 64]);

        % Compute HoG features for the window
        hogFeatures = extractHOGFeatures(newWindow);
        % Z-score normalization
        hogFeatures = (hogFeatures - mean(hogFeatures))/std(hogFeatures);

        % Classify the window
        [label, score] = predict(classifier, hogFeatures);

        % If it's a positive label and the score is higher than current best, store this window
        if label == 1 && score(2) > bestScore
            bestScore = score(2);
            bestWindow = [y, x, newWindowSize(2), newWindowSize(1)]; % [x, y, width, height]
            bestWindowSize = newWindowSize;
        end
    end
end

```

```

        end
    end

function windowSize = findInitialWindow(classifier, I)
    % Initialize window sizes, you may adjust as per your requirements
    window_sizes = [1000, 800, 600, 400, 200, 100];

    maxConfidence = -Inf;
    bestWindowSize = [120, 120]; % default value

    % Step size for the sliding window
    stepSize = [30, 30];

    for idx = 1:length(window_sizes)
        for idy = 1:length(window_sizes)

            currentWindowSize = [window_sizes(idx), window_sizes(idy)];

            for x = 1:stepSize(1):size(I,1) - currentWindowSize(1)
                for y = 1:stepSize(2):size(I,2) - currentWindowSize(2)
                    % Extract window from the image
                    window = I(x:x+currentWindowSize(1)-1, y:y+currentWindowSize(2)-1, :);
                    window = imresize(window, [64, 64]);

                    % Compute HoG features for the window
                    hogFeatures = extractHOGFeatures(window);
                    % Z-score normalization
                    hogFeatures = (hogFeatures - mean(hogFeatures))/std(hogFeatures);

                    % Classify the window and get the score
                    [label, score] = predict(classifier, hogFeatures);

                    if maxConfidence < score(2)
                        maxConfidence = score(2);
                        bestWindowSize = currentWindowSize;
                    end
                end
            end
        end
    end

    % Return the best window size
    windowSize = bestWindowSize;
end

```

```

function [classifier, features, labels] = trainModel(BB, Idir, fpt)
    nf = numel(Idir); % total number of image files

    % Test extraction of HoG features to get feature vector length
    testImage = imread(horzcat(Idir(1).folder,'/',Idir(1).name));
    testFeatures = extractHOGFeatures(imresize(testImage, [64, 64]));
    featureLength = length(testFeatures);

    % Variables to hold features and labels
    % for every image we'll have fpt positive and negative features
    features = zeros(nf * fpt * 2, featureLength);
    labels = zeros(nf * fpt * 2, 1);

    position = 1;
    tic
    for i = 1:nf

        filename = horzcat(Idir(i).folder,'/',Idir(i).name);
        I = imread(filename);

        % Only process if isLost is 0
        if BB(i,6) == 0
            %bbox = [BB(i,3), BB(i,3)+BB(i,5), BB(i,2), BB(i,2)+BB(i,4)];
            bbox = BB(i, 2:5);

            [pos_features, neg_features] = genFeatures(I, bbox, fpt);

            for j = 1:fpt
                % positive ones
                features(position, :) = pos_features(j,:);
                labels(position) = 1;

                % negative ones
                features(position + 1, :) = neg_features(j,:);
                labels(position + 1) = 0;

                position = position + 2;
            end
        end
    end
    toc
    classifier = fitcsvm(features,labels);
    classifier = compact(classifier);
end

function [pos_features, neg_features] = genFeatures(image, bbox, num_features)

```

```

testFeatures = extractHOGFeatures(imresize(image, [64, 64]));
featureLength = length(testFeatures);

pos_features = zeros(num_features, featureLength);
neg_features = zeros(num_features, featureLength);
for i = 1:num_features

    % pos features
    posROI = generateRandomROI(image, bbox, "positive");
    posROI = imresize(posROI, [64, 64]);
    posHogFeatures = extractHOGFeatures(posROI);
    % Z-score normalization
    posHogFeatures = (posHogFeatures - mean(posHogFeatures))/std(posHogFeatures);
    pos_features(i, :) = posHogFeatures;

    % neg features
    negROI = generateRandomROI(image, bbox, "negative");
    negROI = imresize(negROI, [64, 64]);
    negHogFeatures = extractHOGFeatures(negROI);
    % Z-score normalization
    negHogFeatures = (negHogFeatures - mean(negHogFeatures))/std(negHogFeatures);
    neg_features(i, :) = negHogFeatures;
    %imshow(negROI)
end
end

function ROI = generateRandomROI(image, bbox, type)
im_size = size(image);
width = bbox(3);
height = bbox(4);
switch type
case "positive"
    % Restrict the random number generation to the vicinity of the bbox
    lowerX = max(1, bbox(1) - width * 0.2);
    lowerY = max(1, bbox(2) - height * 0.2);

    upperX = min(im_size(2) - width, bbox(1) + width*0.2);
    upperY = min(im_size(1) - height, bbox(2) + height*0.2);

    randX = randi([round(lowerX), round(upperX)]);
    randY = randi([round(lowerY), round(upperY)]);
    % Generate until we find one that intersects >= 0.9
    while bboxOverlapRatio([randX, randY, width, height],bbox) < 0.7
        randX = randi([round(lowerX), round(upperX)]);
        randY = randi([round(lowerY), round(upperY)]);
    end
end

```

```

    end

    case "negative"
        % Restrict the random number generation to the vicinity of the bbox
        lowerX = max(1, bbox(1) - width);
        lowerY = max(1, bbox(2) - height);

        upperX = min(im_size(2) - width, bbox(1) + width);
        upperY = min(im_size(1) - height, bbox(2) + height);

        randX = randi([round(lowerX), round(upperX)]);
        randY = randi([round(lowerY), round(upperY)]);

        % Generate until we find one that intersects <= 0.1
        while bboxOverlapRatio([randX, randY, width, height], bbox) > 0.3
            randX = randi([round(lowerX), round(upperX)]);
            randY = randi([round(lowerY), round(upperY)]);
        end
    end

    ROI = image(randY:randY+height, randX:randX+width, :);
end

```

A.4 Codi Reconeixement Mètode No Propi

Enllaç al Notebook per provar el codi: Notebook

```
from ultralytics import YOLO
import numpy as np
import cv2
from google.colab.patches import cv2_imshow
import glob
import os
import yaml

def setup(dir):
    os.chdir(dir)

    model = YOLO('yolov8n.pt')
    return model

# Yolo expects, instead of a bounding box with x, y, width, length,
# the center of the object and its width and height.
# It also needs everything normalized.

# All images are 1280x720, so we'll use those values to normalize
def preprocess(sequence):

    BB = open(sequence + "/groundtruth_rect.txt", "r+")

    # si no se encuentra en la imagen, eliminar de entrenamiento
    for i, line in enumerate(BB):
        row = line.split(",")

        idx = i + 1

        if (idx < 10):
            name = '0000' + str(idx)

        elif (idx < 100):
            name = '000' + str(idx)
        else:
            name = '00' + str(idx)

        # if 5th element is 1, object is lost. We remove image
        if row[5] == 1:
            # we delete the image and skip the iteration
            os.remove(sequence + '/img/' + name + '.jpg')
            continue
```

```

# Transform and normalize
size_x = 1280
size_y = 720

width = int(row[3]) / size_x
height = int(row[4]) / size_y
x = (int(row[1]) / size_x) + (width/2)
y = (int(row[2]) / size_y) + (height/2)

data = "0 " + str(x) + " " + str(y) + " " + str(width) + " " + str(height)

BB = open(sequence + '/img/' + name + '.txt', 'w+')
BB.write(data)

def genYAMLandTrain(dir, model, sequence):
    data_yaml = dict(
        path = dir + sequence,
        train = 'img', # train image directory
        val = 'img', # validation image directory
        nc = 1, # number of classes
        names = ['object'] # list of class names
    )

    # Write the dictionary to a yaml file
    with open('data.yaml', 'w') as outfile:
        yaml.dump(data_yaml, outfile, default_flow_style=False)

    model.train(data="data.yaml", epochs=3) # train the model
    return model

def bboxOverlayRatio(boxA, boxB, ratiotype):
    """
    Based on:
    https://gist.github.com/meyerjo/dd3533edc97c81258898f60d8978eddc

    """
    # determine the (x, y)-coordinates of the intersection rectangle
    xA = max(boxA[0], boxB[0])
    yA = max(boxA[1], boxB[1])
    xB = min(boxA[2], boxB[2])
    yB = min(boxA[3], boxB[3])

    # compute the area of intersection rectangle
    interArea = abs(max((xB - xA, 0)) * max((yB - yA), 0))

```

```

if interArea == 0:
    return 0
# compute the area of both the prediction and ground-truth
# rectangles
boxAArea = abs((boxA[2] - boxA[0]) * (boxA[3] - boxA[1]))
boxBArea = abs((boxB[2] - boxB[0]) * (boxB[3] - boxB[1]))

match ratiotype:
    case "union":
        # compute the intersection over union by taking the intersection
        # area and dividing it by the sum of prediction + ground-truth
        # areas - the intersection area
        iou = interArea / float(boxAArea + boxBArea - interArea)
    case "min":
        # compute the intersection over union by taking the intersection
        # area and dividing it by the minimum area of the two bounding boxes.
        iou = interArea / float(min(boxAArea, boxBArea))
    case _:
        print('Command not recognized')

# return the intersection over union value
return iou

def calculateAccuracy(sequence_name, model, show):
    BBFile = sequence_name + '/groundtruth_rect.txt'
    image_files = sorted(glob.glob(sequence_name + '/img/*.jpg'))

    correct_predictions = 0
    total_predictions = 0

    # Load ground truth
    BB = np.loadtxt(BBFile, delimiter=',')

    for frame_id, row in enumerate(BB):
        # Check if the object is not lost in this frame
        if row[5] == 0:
            # Get the ground truth bounding box
            ground_truth_bb = [int(v) for v in row[1:5]] # Convert to integers

            # For simplicity, convert to xyxy
            ground_truth_bb[2] = ground_truth_bb[0] + ground_truth_bb[2]
            ground_truth_bb[3] = ground_truth_bb[1] + ground_truth_bb[3]

            # Load image
            img = cv2.imread(image_files[frame_id])
            # Perform object detection

```

```

obj = model(img, verbose=False)

if len(obj[0].boxes) == 0:
    total_predictions +=1
    continue

# Convert tensor to numpy array
predicted_bb = obj[0].boxes.xyxy[0].cpu().numpy()
# Convert to integers
predicted_bb = [int(v) for v in predicted_bb]

if show:
    # Draw ground truth and predicted bounding boxes on the image
    img = cv2.rectangle(img, (ground_truth_bb[0], ground_truth_bb[1]), \
        (ground_truth_bb[2], ground_truth_bb[3]), (0, 255, 0), 2)
    img = cv2.rectangle(img, (predicted_bb[0], predicted_bb[1]), \
        (predicted_bb[2], predicted_bb[3]), (0, 0, 255), 2)

    # Add labels
    cv2.putText(img, 'Ground Truth', (ground_truth_bb[0], ground_truth_bb[1] - 10), \
        cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)
    cv2.putText(img, 'Prediction', (predicted_bb[0], predicted_bb[1] - 10), \
        cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 0, 255), 2)

    # Show the image
    cv2_imshow(img)

# Compare the predicted bounding box with the ground truth.
# You may need to adjust this depending on what constitutes a "correct" prediction.
# Here, I'm considering a prediction correct if the predicted bounding box and
# the ground truth bounding box are the same.
if bboxOverlayRatio(predicted_bb, ground_truth_bb, "min") >= 0.75:
    correct_predictions += 1

total_predictions += 1

# Calculate accuracy as the number of correct predictions divided
# by the total number of predictions
accuracy = correct_predictions / total_predictions if total_predictions > 0 else 0

return accuracy

def main():
    # Change these two:
    dir = '/content/drive/MyDrive/SP_VC/' # exemple
    sequence = 'MotorcycleChase' # exemple

```

```
model = setup(dir)
preprocess(sequence)
model = genYAMLandTrain(dir, model, sequence)
accuracy = calculateAccuracy(sequence, model, False)
print(f'Accuracy: {accuracy}')

if __name__ == '__main__':
    main()
```

B Entrenament dels classificadors

B.1 Mètode Propi

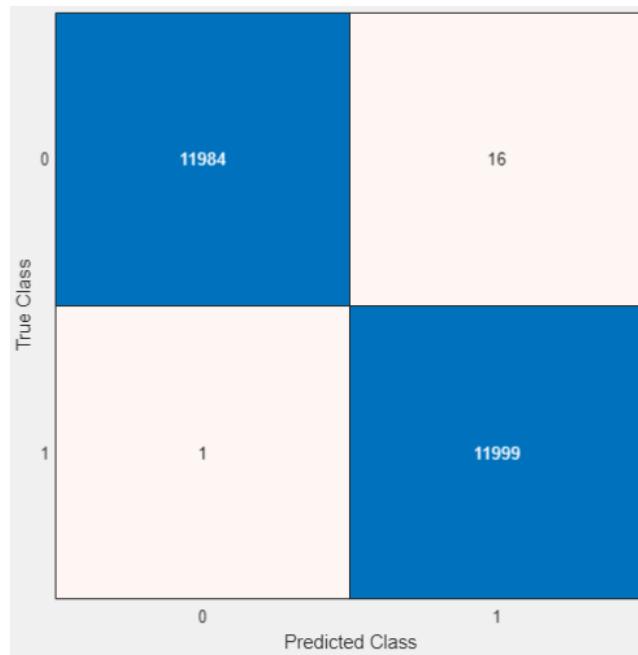


Figura 10: Mètode Propi, Entrenament del classificador per Alladin

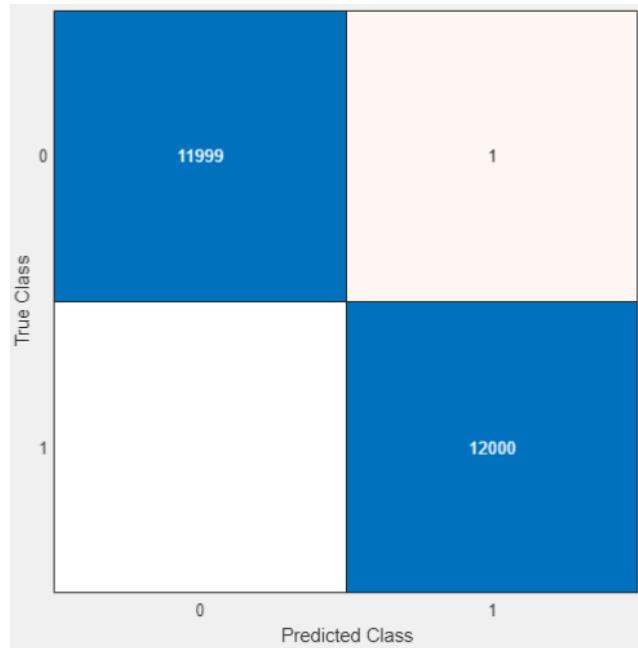


Figura 11: Mètode Propi, Entrenament del classificador per Bike



Figura 12: Mètode Propi, Entrenament del classificador per Boat

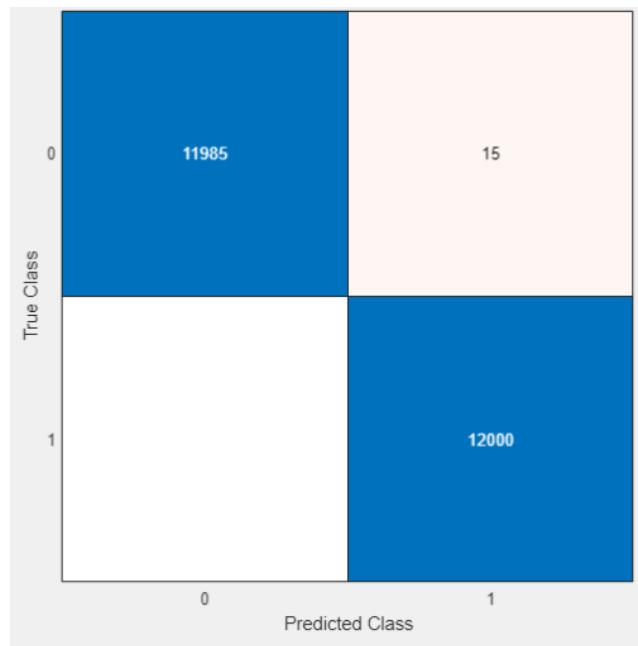


Figura 13: Mètode Propi, Entrenament del classificador per CarChase1

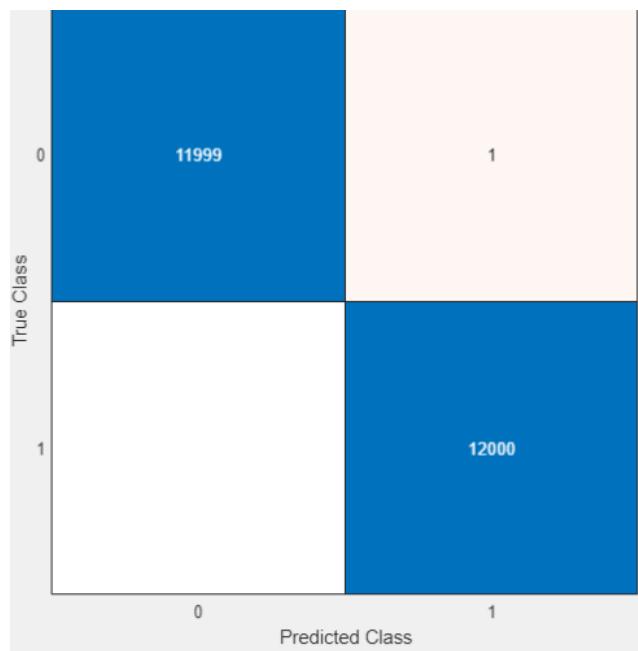


Figura 14: Mètode Propi, Entrenament del classificador per Dashcam

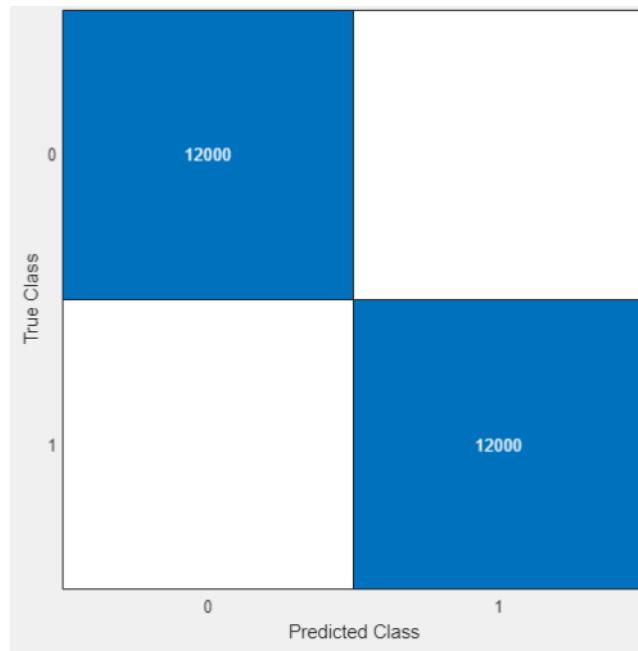


Figura 15: Mètode Propi, Entrenament del classificador per Driftcar2

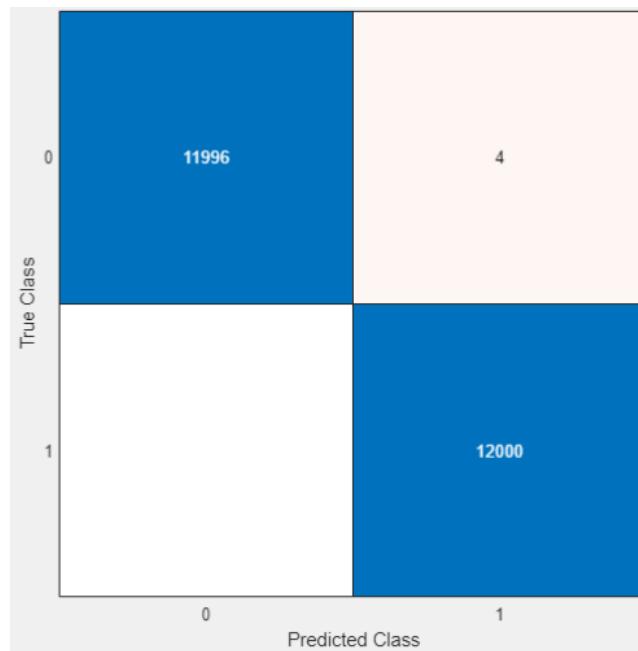


Figura 16: Mètode Propi, Entrenament del classificador per Drone1

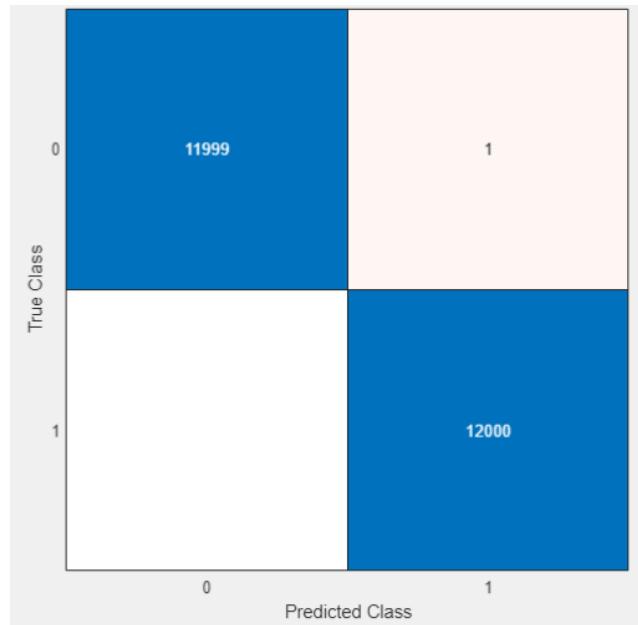


Figura 17: Mètode Propri, Entrenament del classificador per Drone2

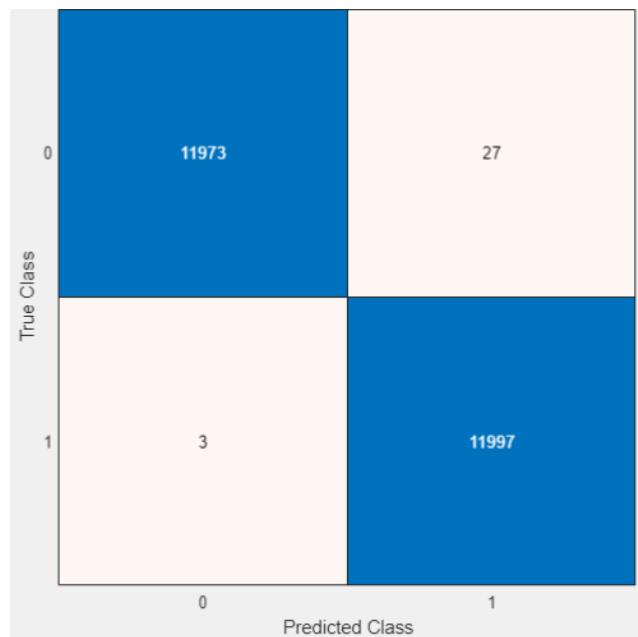


Figura 18: Mètode Propri, Entrenament del classificador per Drone3

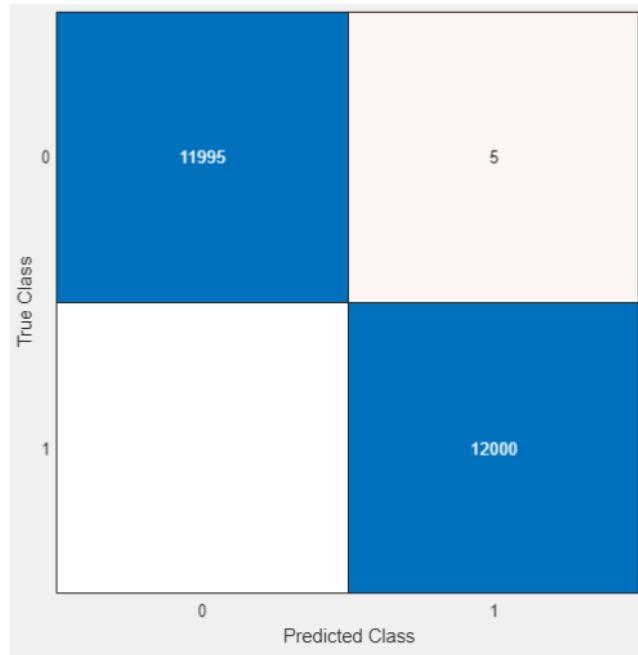


Figura 19: Mètode Propi, Entrenament del classificador per Jet3

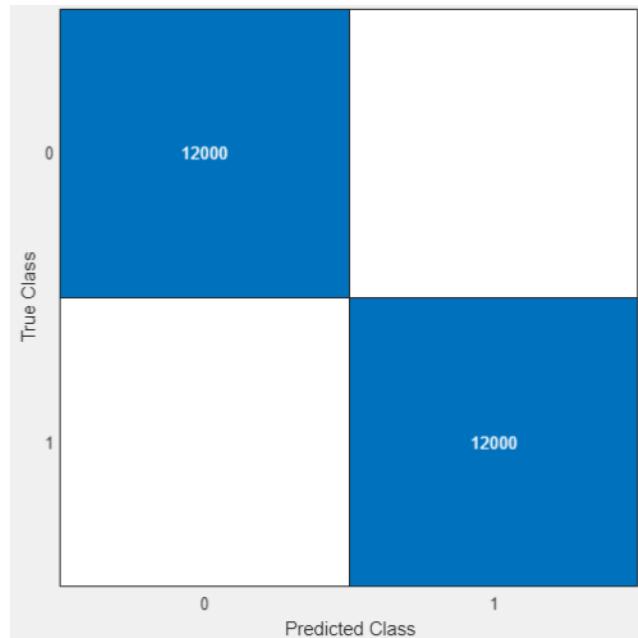


Figura 20: Mètode Propi, Entrenament del classificador per Mohiniyattam

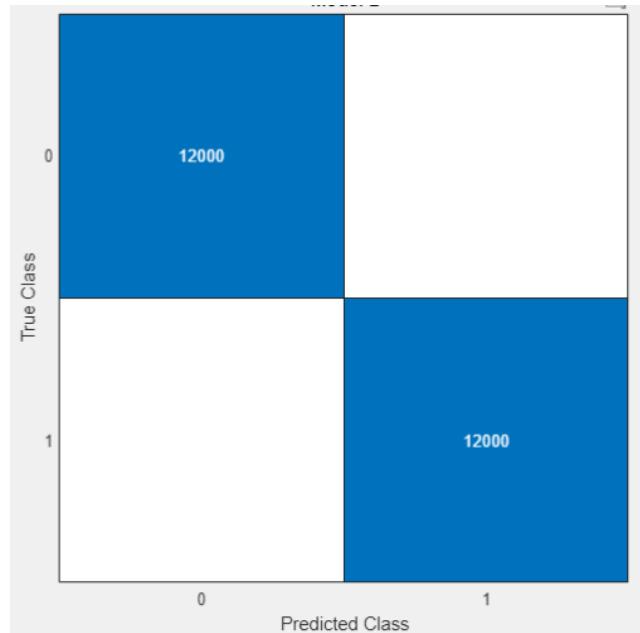


Figura 21: Mètode Propi, Entrenament del classificador per MotorcycleChase

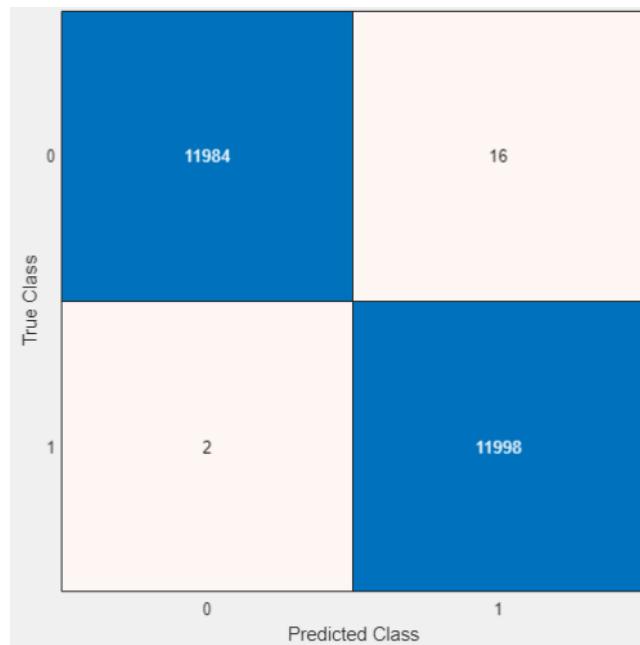


Figura 22: Mètode Propi, Entrenament del classificador per PolarBear1

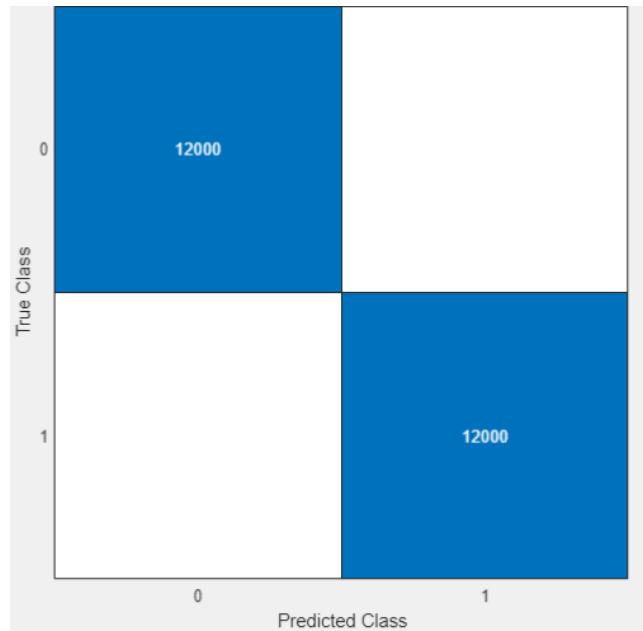


Figura 23: Mètode Propi, Entrenament del classificador per Puppies1

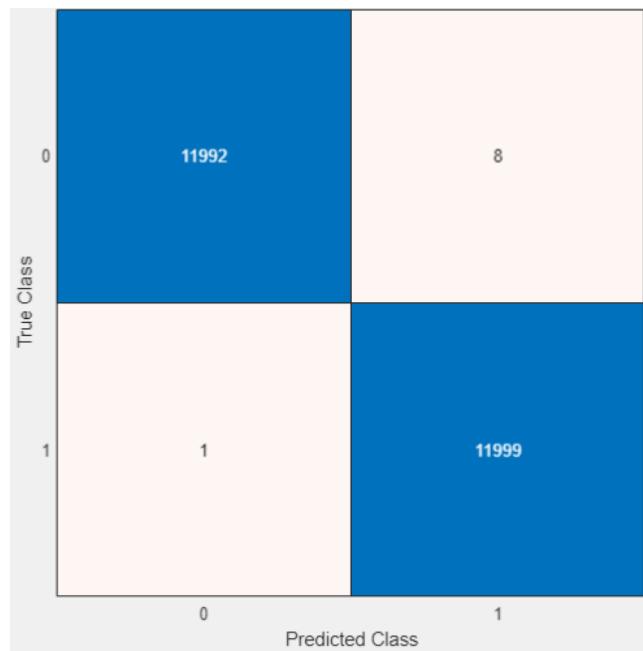


Figura 24: Mètode Propi, Entrenament del classificador per Violinist

B.2 Mètode No Propi

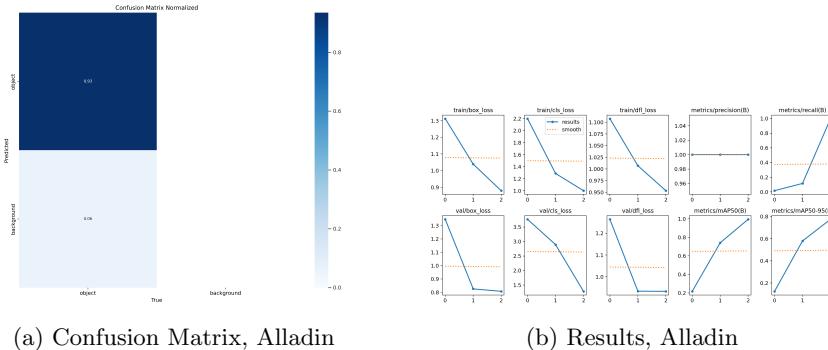


Figura 25: Mètode No Propi, Entrenament del classificador per Alladin

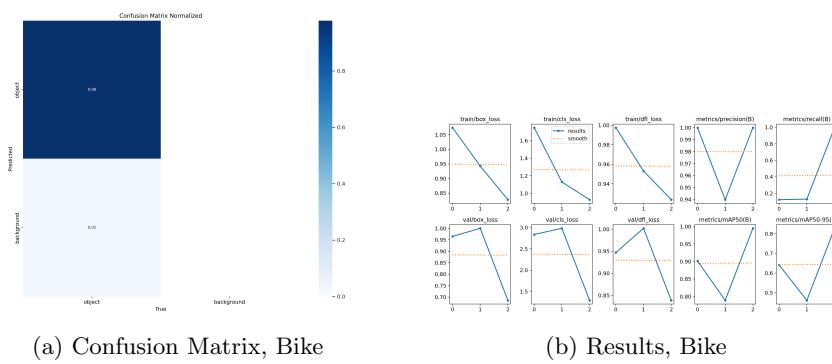


Figura 26: Mètode No Propi, Entrenament del classificador per Bike

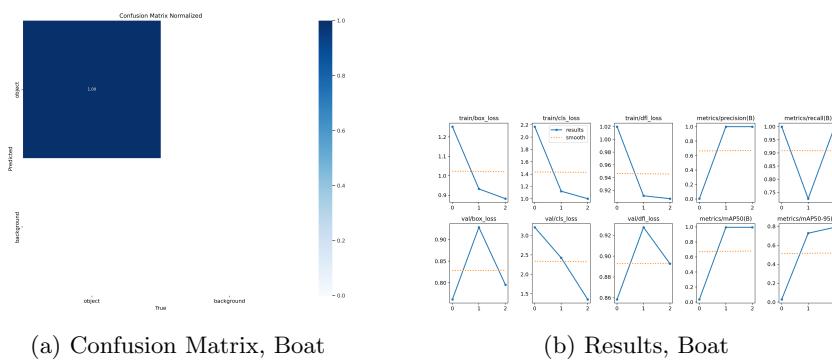
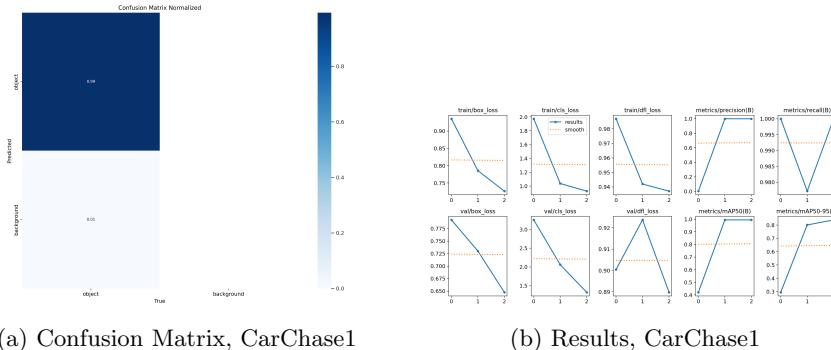


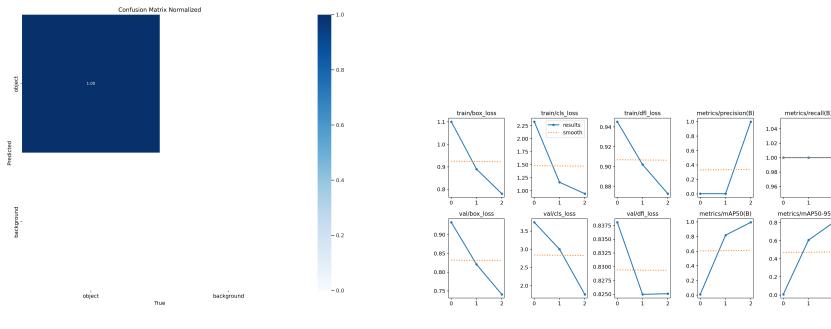
Figura 27: Mètode No Propi, Entrenament del classificador per Boat



(a) Confusion Matrix, CarChase1

(b) Results, CarChase1

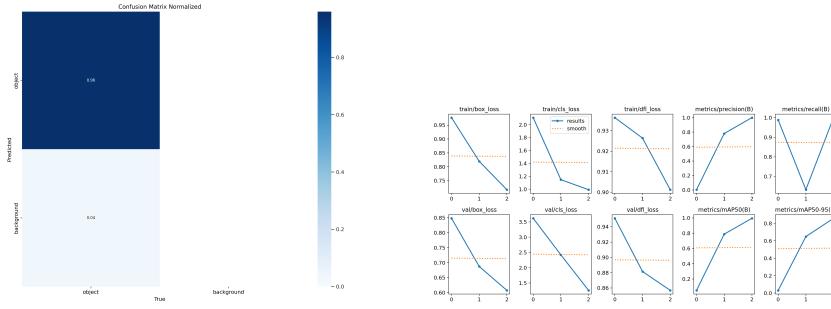
Figura 28: Mètode No Propri, Entrenament del classificador per CarChase1



(a) Confusion Matrix, Dashcam

(b) Results, Dashcam

Figura 29: Mètode No Propri, Entrenament del classificador per Dashcam



(a) Confusion Matrix, Driftcar2

(b) Results, Driftcar2

Figura 30: Mètode No Propri, Entrenament del classificador per Driftcar2

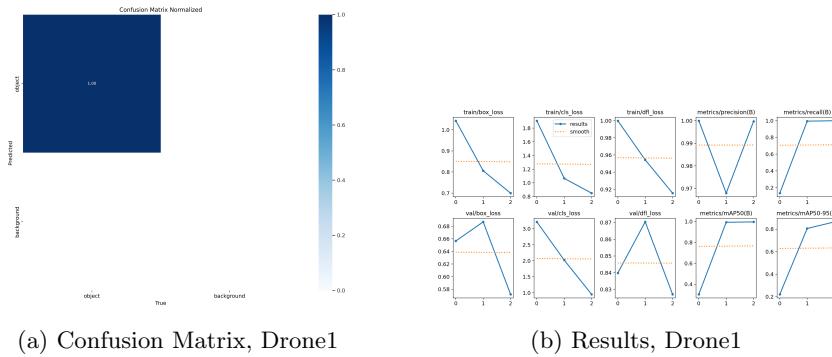


Figura 31: Mètode No Propi, Entrenament del classificador per Drone1

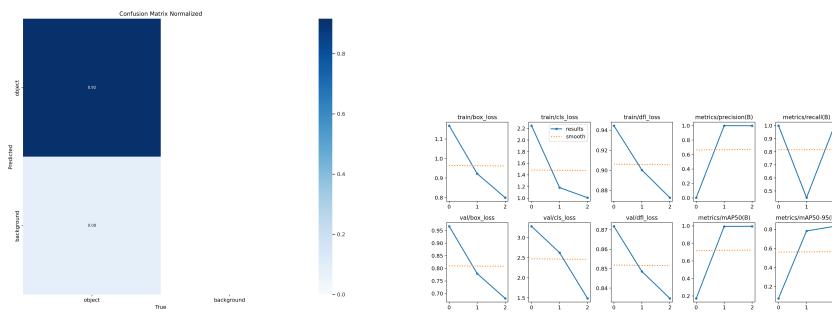


Figura 32: Mètode No Propi, Entrenament del classificador per Drone2

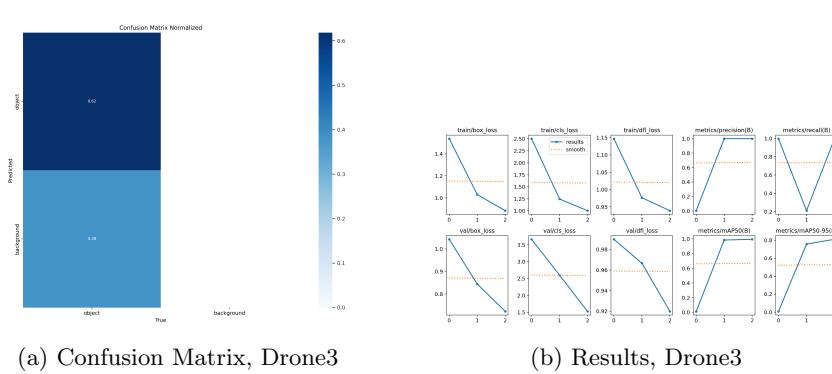


Figura 33: Mètode No Propi, Entrenament del classificador per Drone3

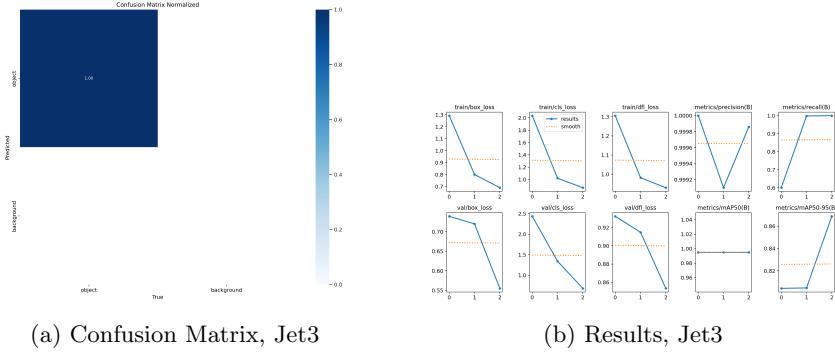


Figura 34: Mètode No Propri, Entrenament del classificador per Jet3

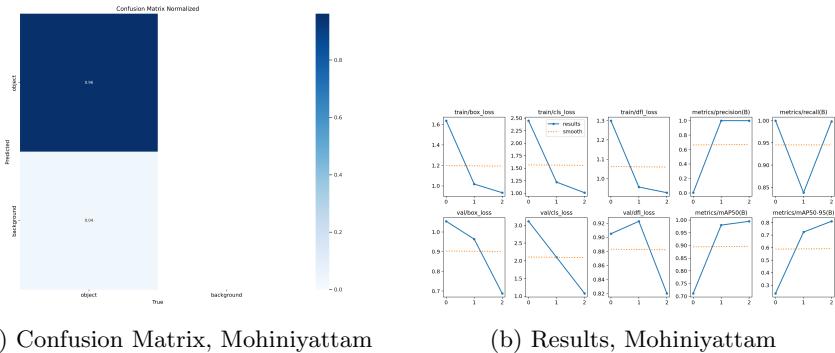


Figura 35: Mètode No Propri, Entrenament del classificador per Mohiniyattam

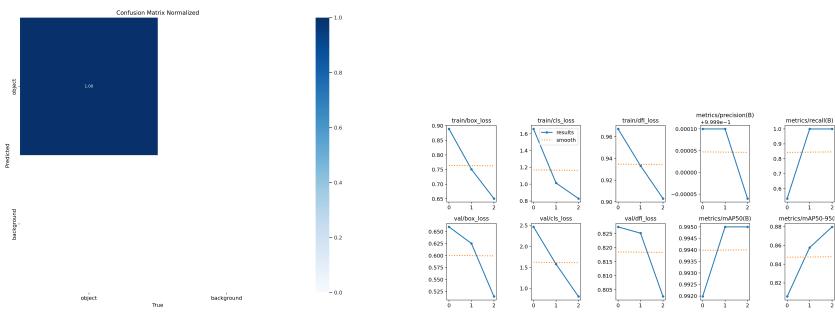
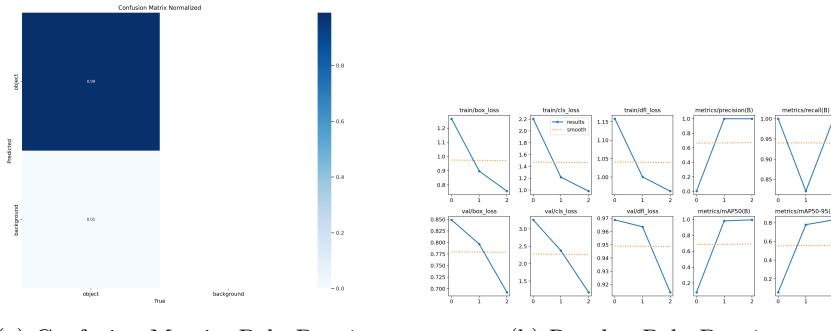


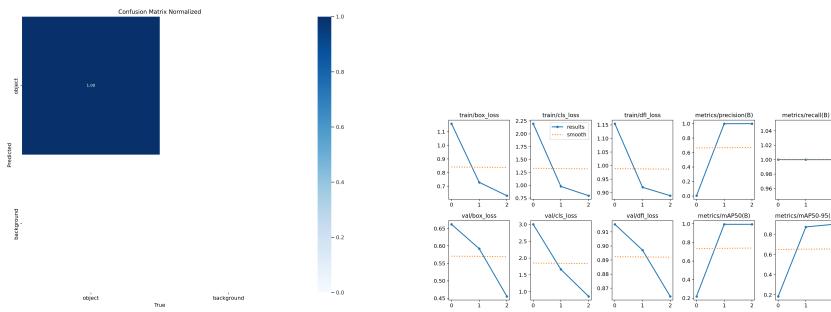
Figura 36: Mètode No Propri, Entrenament del classificador per MotorcycleChase



(a) Confusion Matrix, PolarBear1

(b) Results, PolarBear1

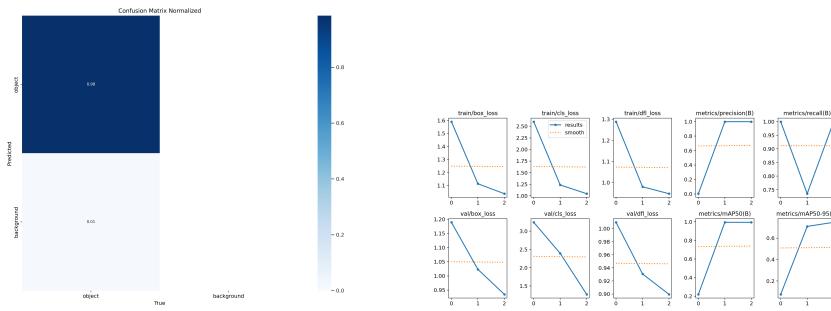
Figura 37: Mètode No Propri, Entrenament del classificador per PolarBear1



(a) Confusion Matrix, Puppies1

(b) Results, Puppies1

Figura 38: Mètode No Propri, Entrenament del classificador per Puppies1



(a) Confusion Matrix, Violinist

(b) Results, Violinist

Figura 39: Mètode No Propri, Entrenament del classificador per Violinist