Deep Learning on Futsal videos

A Degree Thesis

Submitted to the Faculty of the

Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona

Universitat Politècnica de Catalunya

by

Arnau López Garbulosa

In partial fulfilment

of the requirements for the degree in

Grau en Enginyeria de Tecnologies I Serveis de Telecomunicació ENGINEERING

Advisor: Jordi Fornes de Juan

Barcelona, June 2014

Abstract

In futsal, the analysis of matches and trainings is a fundamental part to correct errors and improve aspects of the game, technical and tactical.

This fact has been of sufficient importance for the clubs to incorporate members to the team who are dedicated to collecting and analysing data obtained from the matches. Formerly these members wrote on a sheet important actions such as, for examples, shots made, ball losses, etc. Currently video recording is used for further analysis, which allows the trainer to perform this task himself.

This project consists of a study of artificial intelligence and its branches, investigating the possibilities they offer to apply them to the analysis and obtaining of data from futsal videos. The purpose is to create an application that obtains data and automates the tasks.

Resum

En el futbol sala, l'anàlisi de partits i entrenaments es una part fonamental per corregir errors i millorar aspectes del joc, tant tècnics com tàctics.

Aquest fet ha tingut la importància suficient com per que els clubs incorporessin membres al equip que es dediquin a recollir i analitzar dades obtingudes dels partits. Antigament aquests membres apuntaven a un full accions importants com, per exemple, xuts realitzats, pèrdues de pilota, etc. Actualment s'utilitza la gravació de vídeo per l'anàlisi posterior, fet que permet al mateix entrenador realitzar aquesta tasca.

Aquest projecte consisteix en fer un estudi de la intel·ligència artificial i les seves branques, investigant les possibilitats que ofereixen per aplicar-les a l'anàlisi i obtenció de dades a partir de vídeos de futbol sala. La finalitat consisteix en crear una aplicació que obtingui dades i automatitzar les tasques.

Resumen

En el futbol sala, el análisis de partidos y entrenamientos es una parte fundamental para corregir errores y mejorar aspectos del juego, tanto técnicos como tácticos.

Este hecho ha tenido la importancia suficiente como para que los clubes incorporasen miembros al equipo que se dediquen a recoger i analizar datos obtenidos de los partidos. Antiguamente estos miembros apuntaban en una hoja acciones importantes como, por ejemplo, disparos realizados, pérdidas de balón, etc. Actualmente se utiliza la grabación de video para el análisis posterior, esto permite al mismo entrenador realizar esta tarea.

Este proyecto consiste en hacer un estudio de la inteligencia artificial y sus ramas, investigando las posibilidades que ofrecen para aplicarlas al análisis y obtención de datos a partir de videos de futbol sala. La finalidad consiste en crear una aplicación que obtenga datos i automatizar las tareas.

Agraïments i reconeixements

Voldria agrair al Jordi Fornes De Juan per fer de tutor del projecte, donar-me el seu suport i consells. Donar les gràcies per l'ajuda prestada en l'elecció de la temàtica del treball i per transmetre'm ganes, motivació i confiança per la realització d'aquest.

També agrair a la meva família la confiança que han tingut sempre en mi i recolzar-me en les meves decisions.

Historial de revisions i registre de aprovació

Revisió	Data	Propòsit
1	03/10/2021	Creació
2	04/10/2021	Modificació
3	05/10/2021	Modificació
4	06/10/2021	Modificació
5	08/10/2021	Modificació
6	09/10/2021	Modificació
7	10/10/2021	Modificació
8	10/10/2021	Revisió

LLISTA DE DISTRIBUCIÓ DEL DOCUMENT

Nom	e-mail
Arnau López Garbulosa	arnau1221@gmail.com
Jordi Frones de Juan	jfornes@ac.upc.edu

Escrit per:		Revisat i aprovat per:	
Data	01/10/2021	Data	10/10/2021
Nom	Arnau López Garbulosa	Nom	Jordi Fornes de Juan
Posició	Autor	Posició	Supervisor

Taula de continguts

Llista de figures:

Llista de taules:

1. Introducció

El projecte combina la programació i una de les aficions, el futbol sala, amb l'objectiu d'automatitzar les tasques d'obtenció de dades de partits/entrenaments.

1.1. Motivació

En el futbol sala, l'anàlisi de partits i entrenaments es una part fonamental per corregir errors i millorar aspectes del joc, tant tècnics com tàctics.

Aquest fet ha tingut la importància suficient com per que els clubs incorporessin membres al equip que es dediquin a recollir i analitzar dades obtingudes dels partits. Antigament aquests membres apuntaven a un full accions importants com, per exemple, xuts realitzats, pèrdues de pilota, etc. Actualment s'utilitza la gravació de vídeo per l'anàlisi posterior, fet que permet al mateix entrenador realitzar aquesta tasca.

Després d'haver realitzat un projecte relacionat amb la programació i les bases de dades i el meu interès per seguir formant-me en aquest àmbit, juntament amb el futbol sala com a afició s'han convertit amb la motivació per realitzar aquest projecte.

1.2. Objectius

L'objectiu principal del projecte és investigar les branques de la intel·ligència artificial, concretament el Deep Learning, per desenvolupar aplicacions que permetin l'obtenció de dades a partir de vídeos de futbol sala.

Aquest objectiu principal es podria dividir en els següents objectius:

- Investigar i estudiar les àrees de la intel·ligència artificial (Deep Learning)
- Aprendre un llenguatge de programació nou
- Ampliar els coneixements en programació i bases de dades
- Realitzar una aplicació basada en Deep Learning capaç de extreure informació de vídeos de futbol sala.

1.3. Requeriments i especificacions

Requeriments:

- Ordinador
- Editor de cdi: Sublime Text
- Llenguatge de programació: Python
- Diverses llibreries: (anomenades més endevant)

Especificacions:

- Capacitat de extreure dades a partir de vídeos de futbol sala

1.4. Mètodes i procediments

La realització d'aquest treball comença de zero amb la investigació en un camp desconegut per mi. També es cert que durant el transcurs del projecte i davant una sèrie de dificultats s'acaba utilitzant uns models pre-entrenats (yolov3, MobileNetSSD) per desenvolupar la aplicació.

En quant a la metodologia, s'ha fet un plantejament dels objectius i planificació prèvia, una revisió per observar el progrés i una valoració final.

Per el desenvolupament de l'aplicació s'ha utilitzat algunes característiques típiques de la les metodologies Agile, amb la idea de crear un primer mòdul funcional i anar ampliant el projecte amb nous mòduls a mesura que es va avançant.

1.5. Pla de treball

Per al correcte desenvolupament del projecte, s'han distribuït les tasques d'aquest en sis principals paquets de treball:

1. Proposta de projecte i pla de treball
2. Aprenentatge previ
3. Desenvolupament
4. Revisió crítica
5. Memòria final
6. Presentació

La distribució d'aquests paquets s'ha realitzat com s'indica en el següent diagrama de Gantt.

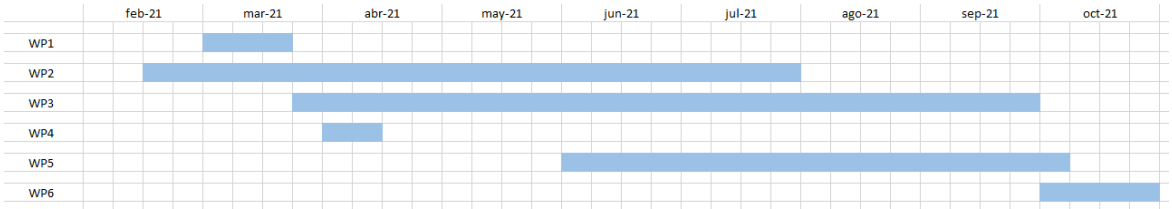


Fig.1 – Diagrama de Gantt del projecte

1.6. Desviacions del pla inicial i incidències

La inexperiència i desconeixença en l'àrea de la intel·ligència artificial ha provocat modificacions tant en els objectius com en la planificació del projecte.

En relació a la planificació les principals desviacions es centren en modificacions de les dates previstes. Al tractar-se d'un àmbit desconegut la recerca de informació i aprenentatge previ ha estat present al llarg de gairebé tot el treball, endarrerint la data d'inici del desenvolupament i entrega del projecte.

Això també a desencadenat en reduir alguns objectius, com per exemple, eliminar la part de les bases de dades, que consistia en ampliar els coneixements en aquesta àrea per

acabar creant i guardant les dades extretes en una. O la creació d'una interfície visual atractiva per a agrupar les diverses aplicacions desenvolupades.

2. Estat de l'art:

2.1. Intel·ligència artificial

Entenem per intel·ligència artificial (IA) com la ciència i enginyeria de crear màquines intel·ligents capaces de imitar les capacitats de resolució de problemes i presa de decisions típiques de la ment humana.

Actualment existeixen una gran quantitat d'aplicacions de IA, entre elles:

- Transcripció veu-text
- Motors de recomanacions
- Bots i assistents
- Conducció automàtica

2.2. Machine Learning

El Machine Learning o aprenentatge automàtic és una àrea de la intel·ligència artificial, que proporciona la capacitat d'aprendre als ordinadors sense la necessitat de ser explícitament programats.

Consisteix en desenvolupar un algoritme de predicció per a cada problema, que aprengui d'un conjunt de dades, amb la finalitat de trobar patrons i construir un model per predir i classificar elements.

Es destaquen tres grans categories:

- **Aprenentatge supervisat**: les dades contenen etiquetes amb la solució, es necessita de la intervenció humana per al etiquetat.
- **Aprenentatge no supervisat**: les dades no inclouen etiquetes, detecta patrons en les dades y els agrupa per qualsevol característica distintiva.
- **Aprenentatge per reforç**: el model aprèn a ser més precís basant-se en la retroalimentació mitjançant recompenses i penalitzacions.

Terminologia

Nom	Descripció
Etiqueta o label	El que intentem predir
Caracteristica, variable o feature	Variable de entrada
Pes o weight	Paràmetre a aprendre
Biaix o bias	Paràmetre a aprendre
Error o loss	Penalització de una mala predicció. Es vol minimitzar.

Model	Relació entre variables, consta de dos fases: entrenament i predicció
Aprenentatge, entrenament o training	S'entrena el model mitjançant un algoritme, ajustant els paràmetres per minimitzar l'error.
Avaluació o validació	Procés en el que s'avalua el model per corregir els seus paràmetres mitjançant la modificació de híper-paràmetres.
Predicció, inferència o inference	Aplicar el model per predir exemples.
Sobre-ajustament o overfitting	Quan el model s'ajusta tant als exemples no es capaç de predir-ne nous.

Taula 1 – Terminologia de Machine Learning

Es pot definir un model com una funció:

$$y = wx + b$$
$$y\text{: } label,\ \ x\text{: } feature,\ \ w\text{: } weight\ o\ peso,\ \ b\text{: } bias\ o\ sesgo$$

2.3. Xarxes neuronals artificials i Deep Learning

El Deep Learning o aprenentatge profund es un subconjunt del Machine Learning. Es basa en una xarxa neuronal artificial amb múltiples capes que contenen diferents números de neurones. Aquestes estan interconnectades entre elles de diverses formes amb les demés capes.

A diferencia dels algoritmes de Machine Learning, els algoritmes de Deep Learning poden determinar quines característiques son les més importants per distingir cada element.

Un model esta composat per una capa d'entrada que rebrà les dades per el processament, múltiples capes ocultes i la capa de sortida que realitza la predicció final.

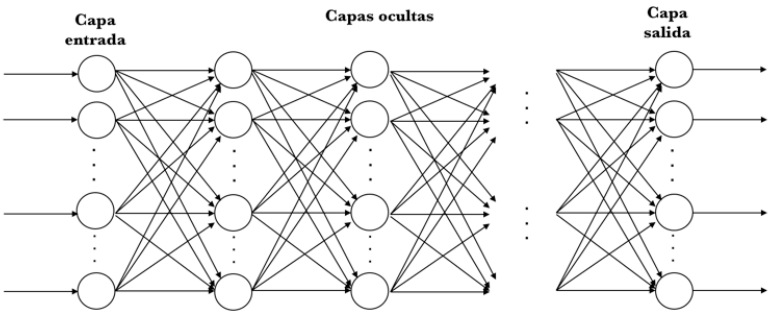


Fig. 2 – Estructura d'un model / xarxa neuronal artificial Deep Learning

Cada neurona té els seus propis paràmetres (pes i biaix) que realitzen una transformació simple de les dades rebudes. La unió de aquestes permet descobrir patrons mes complexes que ens serviran per la predicció.

2.3.1. Tipus de xarxes neuronals

Existeixen diversos tipus de xarxes neuronals com per exemple:

- **Xarxes neuronals convolucionals (CNN)**
  Utilitzades principalment en aplicacions de visió artificial i classificació d'imatges. Poden detectar característiques i patrons dins una imatge permetent tasques com detecció i reconeixement d'objectes.

- **Xarxes neuronals recurrents (RNN)**:
  Utilitzades principalment en aplicacions de llenguatge natural i reconeixement de veu. Aprofiten dades seqüencials o series temporals.

2.3.2. Procés d'aprenentatge d'una xarxa neuronal

El procés d'aprenentatge consisteix en començar amb uns valors de paràmetres petits i aleatoris, per evitar que dues neurones amb els mateixos pesos aprenguin la mateixa característica. Agafar dades de entrada, passar-los per la xarxa, comparar la predicció amb les etiquetes (en el cas de entrenament supervisat) i calcular l'error. Propagar aquest error cap als paràmetres i actualitzar-los. Finalment continuar iterant per anar millorant mica en mica el model.

Existeixen diverses fases en el entrenament d'una xarxa neuronal.

- **Forward propagation**: Aplica les dades de entrada i realitza els càlculs per obtenir una predicció. Es propaga la informació mitjançant la funció d'activació.
- **Estimació de l'error**: es calcula l'error mitjançant la funció de los.
- **Backward propagation**: s'estima l'error, s'avalua el model i es propaga aquesta informació a totes les neurones de la xarxa per ajustar els pesos i biaixos. Per fer aquest reajustament es pot utilitzar l'algoritme gradient descent.

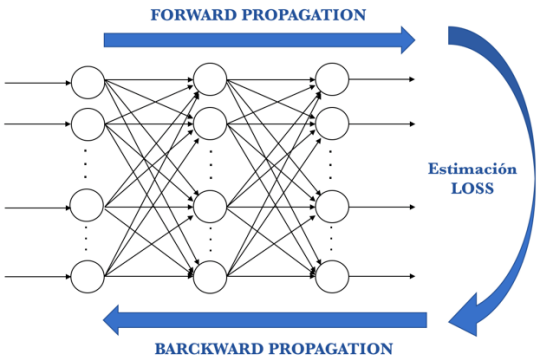


Fig. 3 – Fases de l'entrenament d'una xarxa neuronal artificial.

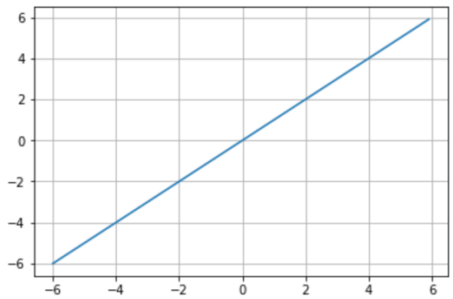
La actualització dels paràmetres es pot realitzar amb diferents freqüències:

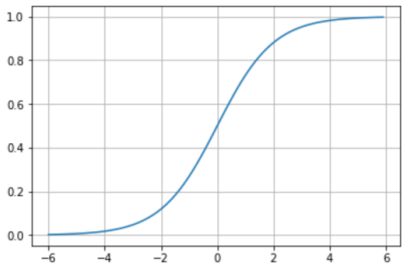
- En cada exemple d'entrada (**Online Learning**)
- En cada subconjunt de dades d'entrada (**Batch Learning**)

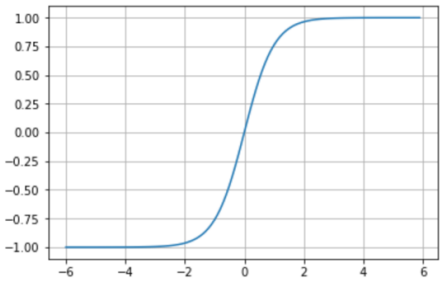
2.3.2.1. Funció d'activació

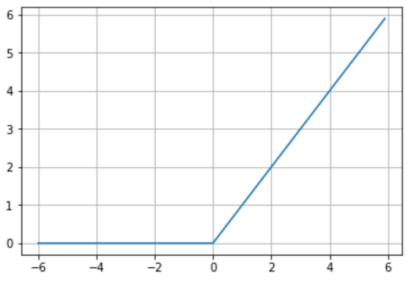
Es tracta d'un filtre a la sortida de la neurona que, modifica el valor o estableix un llindar mínim que s'ha de superar per propagar-se a una altre neurona. Introdueix la no linealitat en las capacitats del modelat.

Algunes de les més freqüents són:

<table>
<tr>
<td> 

Linear
La senyal no cambia</td>
<td> 

Sigmoid
Redueix els extrems. Es passa de rangs infinits al rang (0,1)</td>
</tr>
<tr>
<td> 

Tanh
Relació entre el sinus i el cosinus hiperbòlic. Es passa de rangs infinits al rang (0,1)</td>
<td>$$Softmax_i \frac{e^{evidencia_i}}{\sum_j e^{evidencia_j}}$$
Softmax
Retorna la distribució de probabilitat.</td>
</tr>
<tr>
<td colspan="2"> 

ReLU
Activa un node si la entrada supera un l'ombrall.</td>
</tr>
</table>

Taula 2 – Principals funcions d'activació.

2.3.2.2. Funció de loss

La funció de loss es un paràmetre que permet determinar com de a prop està una xarxa neuronal de ser ideal durant el procés d'entrenament.

Per poder escollir una bona funció de loss es necessari entendre quin tipus d'error es o no acceptable per a un determinat problema.

2.3.2.3. Gradient descent

El descens per gradient permet, mitjançant petits increments amb la ajuda de la derivada de la funció de loss, veure en quina direcció descendir fins al mínim global per tal de minimitzar l'error.

Consisteix en encadenar les derivades de cada capa amb la de la seva superior incorporant la funció d'activació. Després s'actualitzen els paràmetres en sentit contrari al gradient, ja que aquest apunta sempre on s'incrementa la funció.

Per determinar el següent valor es modifica el pes afegint una quantitat proporcional a aquest. La magnitud d'aquest canvi es determina per el valor de gradient i un híper-paràmetre (learning rate)

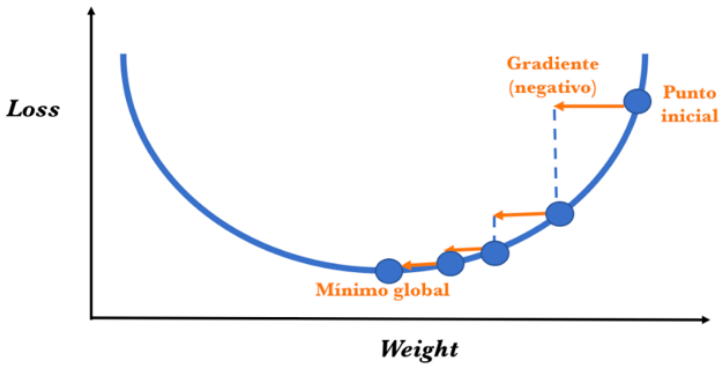


Fig. 4 – Funcionament del gradient descent

2.3.2.4. Paràmetres i híper-paràmetres

Els paràmetres són variables internes del model estimades a partir de les dades com el pes o biaix. Els híper-paràmetres són variables externes del model especificats per el programador.

Numero de epochs

Indica el numero de vegades que totes les dades d'entrenament han passat per la xarxa neuronal durant el procés d'entrenament.

Batch size

És l'argument que indica la mida que tindran els grups de dades que es passaran per la xarxa a cada iteració.

Learning rate

El learning rate és l'escalar per el qual es multiplica la magnitud del gradient en els algoritmes de gradient descent per determinar el següent punt.

El valor més adequat varia en funció del problema. Generalment un valor gran podria indicar un avanç ràpid però per contra podria saltar-se el mínim i anar rebotant sense aproximar-se a aquest, mentre que un valor petit convertiria l'aprenentatge en un procés lent.

Learning rate decay

Aquest híper-paràmetre és una solució al problema del learning rate i s'utilitza per disminuir el learning rate a mida que passen els epochs permetent que l'entrenament avanci més ràpid al principi i més lent a mida que es va apropant al mínim.

Momentum

És una constant entre 0 i 1 que s'utilitza per ponderar els anteriors gradients per evitar aturar-se en un mínim local.

2.3.3. Procés d'avaluació

El procés de validació serveix per fer una avaluació del entrenament i veure si aquest s'ha adaptat a les dades d'entrenament, i així, per poder corregir els híper-paràmetres i fer més eficient el model.

Per veure com de bé es comporta el model amb dades no vistes anteriorment es calcula la loss i la accuracy.

La matriu de confusió es una eina per avaluar models. Consisteix en una taula on es mostren tots els casos possibles de deteccions correctes i incorrectes. L'exemple més simple que trobem és el següent:

		Predicció	
		Positius	**Negatius**
Observació	**Positius**	Vertaders positius (VP)	Falsos Negatius (FN)
	Negatius	Falsos Positius (FP)	Vertaders Negatius (VN)

Taula 3 – Taula d'exemple de la matriu de confusió

Tenim comptabilitzats:

1. Vertaders positius: positius classificats correctament
2. Vertaders negatius: negatius classificats correctament
3. Falsos negatius: negatius classificats incorrectament
4. Falsos positius: positius calculats incorrectament

La accuracy en indica el percentatge de dades, no vistes anteriorment, que ha classificat correctament. Es podria calcular de la següent manera:

$$accuracy = \frac{(VP + VN)}{(VP + FP + VN + FN)}$$

S'ha de tenir en compte que no es diferencia entre falsos positius i negatius podent ser algun d'aquests més perjudicial que l'altre.

Una altre mètrica és la sensibilitat que indica el funcionament en falsos negatius.

$$sensitivity = \frac{VP}{(VP + FN)}$$

2.3.4. Xarxes neuronals convolucionals

Com s'ha comentat anteriorment les xarxes neuronals convolucionals (CNN) son molt utilitzades en visió computacional.

Una de les principals diferencies es que s'assumeix que les dades d'entrada són imatges. Això permet codificar certes propietats per reconèixer elements concrets, de manera que en cada capa la xarxa va aprenent diferents nivells d'abstracció per identificar estructures cada cop més complexes.

Per exemple en el cas de detectar una cara el fet de tenir múltiples capes ens permetrà detectar primer els patrons que identifiquen els elements d'una cara (nas, ulls, orelles, boca, etc.)

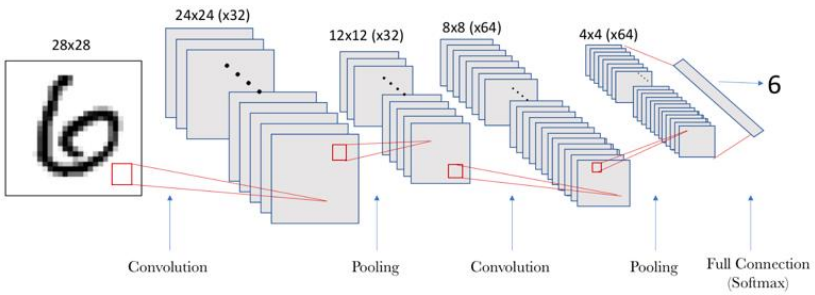


Fig. 5 – Representació d'una xarxa neuronal convolucional

2.3.4.1. Components bàsics

Operació de convolució

Les capes convolucionals a diferencia de les densament connectades, aprenen patrons locals en petites finestres de dos dimensions.

El principal propòsit es detectar característiques visuals en imatges com poden ser arestes, línies, etc. Aquesta propietat permet que una vegada apresa una característica reconèixer-la a qualsevol part.

Les capes convolucionals poden aprendre jerarquies espacials de patrons preservant relacions espacials. Per exemple, una primera capa pot aprendre elements bàsics com arestes, la segona patrons composats per elements bàsics apresos a la capa anterior i així anar escalant.

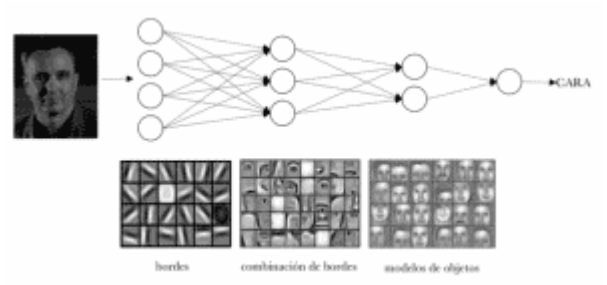


Fig. 6 – Esquema de les jerarquies espacials que poden aprendre les xarxes neuronals convolucionals.

Aquestes capes operen sobre tensors 3D essent els eixos la altura, amplitud i canal o profunditat.

El funcionament consisteix en utilitzar una finestra que introduirà les dades d'entrada d'una neurona i desplaçar-la per obtenir la resta. Es poden utilitzar diverses longituds de desplaçament (stride) o omplir amb valors el voltant de les imatges (padding).

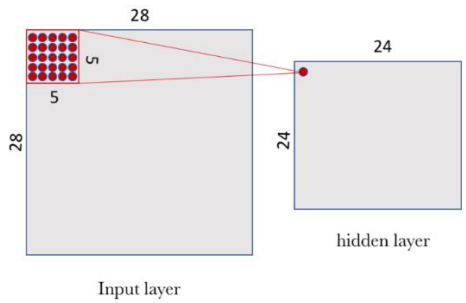


Fig. 7 – Funcionament de la finestra en xarxes neuronals convolucionals

Els paràmetres que s'utilitzen (matriu de pesos i biaix) poden ser els mateixos per a totes les neurones de la mateixa capa. D'aquesta forma es redueix el nombre de paràmetres a ajustar d'una xarxa. Però això significa que només es podrà detectar una característica en cada capa. Per aquest motiu una capa convolucional completa en una xarxa neuronal convolucional inclou varis filtres.

Operació de pooling

Les capes de pooling s'utilitzen immediatament després de les convolucionals i simplifiquen la informació mantenint la relació espacial creant una versió condensada.

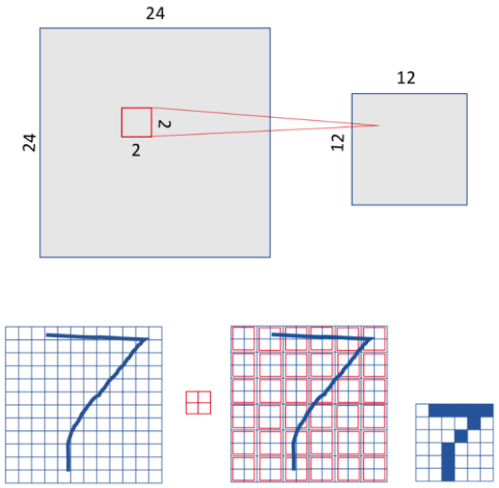


Fig. 8 – Funcionament de l'operació de pooling

Existeixen varies formes de condensar la informació, entre elles:

- **Max-pooling**: es queda amb el valor màxim de la finestra d'entrada.
- **Average-pooling**: valor mitjà del grup de punts.

La capa de pooling contindrà tants filtres com tingui la capa convolucional.

2.3.4.2. Hiperparàmetres

Mida y número de filtres

El numero de filtre indica el numero de característiques que volem manejar, alguns valors estàndards són 32 o 64 i la mida entre 3x3 i 5x5.

Padding

Si es vol obtenir una imatge a la sortida de les mateixes dimensions que l'entrada es pot utilitzar l'híper-paràmetre padding per agregar zeros al voltant de les imatges abans de aplicar la finestra.

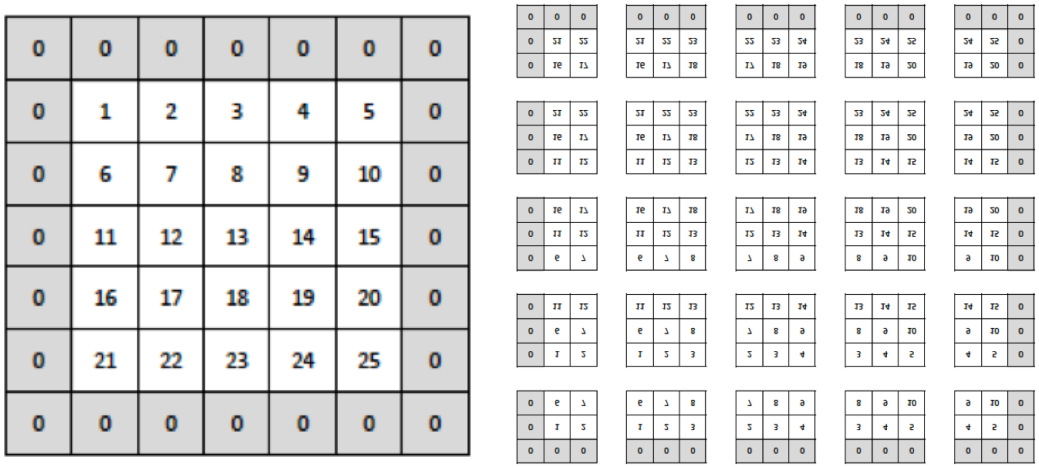


Fig. 9 – Exemple de padding

Stride

Indica el número de de passos que es desplaça la finestra del filtres, per a valors grans es reduirà més la mida de la informació que passarà a la següent capa.

2.3.5. Problemàtiques en el entrenament de models

2.3.5.1. Overfitting

L'overfitting succeeix quan s'aprenen detalls que no son generals, incloent els defectes o soroll. Això provoca que només es considerin vàlides les dades idèntiques utilitzades a l'entrenament.

Aquest fet pot venir donat per dues causes principals, un conjunt de dades molt reduït o realitzar moltes iteracions amb les mateixes dades.

Per reduir aquest fenomen s'utilitzen les dades de validació, número de epochs, learning rate, etc. Conceptes explicats anteriorment.

Data augmentation

Per reduir-lo existeix també una tècnica anomenada data augmentation, , que consisteix en generar dades d'entrenament a partir de les disponibles, realitzant transformacions aleatòries sobre aquestes. En el cas de les imatges algunes opcions serien rotar, voltejar o traslladar-les, sempre tenint en compte de no generar imatges que no es trobarien en la realitat per tal de no empitjorar el model.

Transfer Learning

El transfer learning és una altre tècnica per reduir l'overfitting i consisteix en utilitzar les característiques apreses d'un model pre-entrenat per tal de millorar-lo.

Existeixen dues formes de fer-ho:

- **Feature extraction**: consisteix en reutilitzar la base convolucional amb les característiques apreses i substituir el classificador final.
- **Fine tuning**: consisteix en entrenar també algunes capes finals i entrenar-la juntament amb el classificador

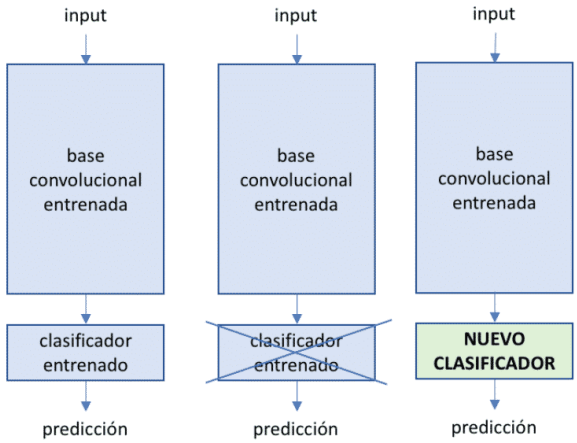


Fig. 10 – Esquema de la tècnica de transfer learning: feature extraction.

2.3.6. Xarxes neuronals recurrents

Les xarxes neuronals recurrents o RNN són una classe de xarxes que permeten analitzar dades de series temporals permetent tractar la dimensió del temps.

2.3.6.1. Conceptes bàsics

Neurona recurrent

A diferencia de les xarxes convulucionals en les recurrents la funció d'activació no nomes actua en la direcció d'entrada a sortida, sinó que també ho fa cap endarrere. S'inclou una retroalimentació entre neurones dins les capes.

Utilitzant aquesta idea es pot alimentar una neurona amb la entrada corresponent a la capa anterior i la sortida del instant previ de la pròpia capa.

Memory cell

Entenent aquest concepte es pot dir que una neurona recurrent té en certa forma memòria. La part de la xarxa que preserva un estat a través del temps s'anomena memory cell o cell.

Long-Short Term Memory

Las Long-Short Term Memory o LSTM són una extensió de les xarxes recurrents que amplien la seva memòria per aprendre de experiències importants que han passat fa estona, permetent recordar les seves entrades durant un temps. Les seves neurones poden llegir, escriure i esborrar informació de la seva memòria.

A traves del pesos s'assigna una importància a la informació que passa per aquella neurona i amb això es decideix si es guarda informació o no a memòria.

2.4. Implementació de models en Keras

2.4.1. Tensorflow, Keras i opencv

Per a la realització del treball s'han escollit tres llibreries que s'utilitzen per l'entrenament i desenvolupament de models en Machine Learning. L'elecció d'aquestes ha estat per la corba d'aprenentatge suau que presenten, la popularitat, per tant més informació disponible, i contribucions d'altres. També per la simplicitat i llegibilitat que mostren i que s'utilitzen en el llenguatje de programació python que també compleix els aspectes anteriors.

2.4.2. Xarxes densament connectades en Keras

La estructura principal de la llibreria Keras es la classe Squential que permet crear una xarxa neuronal bàsica i el mètode add per afegir capes. Només es necessari indicar la forma dels tensors a la primera capa.

A l'hora d'afegir-les s'indicarà el tipo de capa, en aquest cas Dense.

```
from keras.models import Sequential
from keras.layers.core import Dense, Activation

model = Sequential()
model.add(Dense(10, activation='sigmoid', input_shape=(784,)))
model.add(Dense(10, activation='softmax'))
```

Es configura el procés de aprenentatge mitjançant el mètode compile definint com a paràmetres:

- Funció de los: per avaluar el grau d'error
- Optimitzador: algoritme que permet calcular els pes.
- Mètrica: monitoritza el procés.

```
model.compile(loss="categorical_crossentropy", optimizer="sgd",
metrics = ['accuracy'])
```

Per entrenar el model s'utilitza el mètode fit amb els següents paràmetres:

- Dades de entrada
- Etiquetes de les dades

- Batch size: número de dades utilitzades a cada iteració
- Epochs: número de vegades que s'utilitzaran totes les dades

```
model.fit(x_train, y_train, batch_size=100, epochs=5)
```

La funció de avaluació retorna la loss i la accuracy

```
test_loss, test_acc = model.evaluate(x_test, y_test)
```

Finalment per realitzar predicció utilitzant la funció predict se li passen les dades i retorna el resultat estimat.

```
predictions = model.predict(x_test)
```

2.4.3. Implementació d'una xarxa neuronal convolucional

Per crear un xarxa neuronal convolucional caldrà anar intercalant capes convolucionals (Conv2D) amb capes de pooling (MaxPooling2D) afegint una capa densament connectada al final.

```
model.add(layers.Conv2D(32,(5,5),activation='relu',input_shape=(28
,28,1)))
model.add(layers.MaxPooling2D((2,2)))

model.add(layers.Flatten())
model.add(layers.Dense(10, activation='softmax'))
```

2.5. Computer vision

La visió per computació és una àrea important i que molts cops està lligada al Deep Learning, com per exemple en aquest projecte en que es vol extreure informació a partir d'imatges o vídeos.

Es tracta d'un camp de la intel·ligència artificial que permet als ordinadors y sistemes derivar informació significativa de imatges digitals o vídeos per pendre mesures o realitzar recomanacions basades en aquesta.

La visió per computació entrena a les maquines per realitzar aquestes funcions utilitzant càmeres, dades i algoritmes. Per aconseguir aquest objectius s'ajuda del Deep Learning utilitzant les xarxes convolucionals.

3. Metodologia i desenvolupament del projecte

3.1. Instal·lació de l'entorn

Les eines i programes utilitzats per al desenvolupament d'aquest projecte han estat:

- **Ordinador** amb el sistema operatiu Windows 10
- **Python** com a llenguatge per a programar l'aplicació
- **Smartphone** per a la gravació
- **DroidCamApp** com a aplicació per connectar el telèfon mòbil al ordinador
- L'editor de codi **Sublime Text**
- **Entorn virtual i llibreries**:

```
absl-py==0.13.0
astunparse==1.6.3
cachetools==4.2.2
certifi==2021.5.30
charset-normalizer==2.0.4
clang==5.0
cmake==3.21.1.post1
dlib==19.22.0
flatbuffers==1.12
gast==0.4.0
google-auth==1.35.0
google-auth-oauthlib==0.4.6
google-pasta==0.2.0
grpcio==1.40.0
h5py==3.1.0
idna==3.2
imutils==0.5.4
keras==2.6.0
Keras-Preprocessing==1.1.2
Markdown==3.3.4
numpy==1.19.5
```

```
oauthlib==3.1.1
opencv-contrib-python==4.5.2.54
opt-einsum==3.3.0
protobuf==3.17.3
pyasn1==0.4.8
pyasn1-modules==0.2.8
requests==2.26.0
requests-oauthlib==1.3.0
rsa==4.7.2
scipy==1.7.1
six==1.15.0
tensorboard==2.6.0
tensorboard-data-server==0.6.1
tensorboard-plugin-wit==1.8.0
tensorflow-cpu==2.6.0
tensorflow-estimator==2.6.0
termcolor==1.1.0
typing-extensions==3.7.4.3
urllib3==1.26.6
Werkzeug==2.0.1
wrapt==1.12.1
```

Fig. 11 – Llibreries instal·lades a l'entorn virtual

3.2. Plantejament inicial

En un principi, la idea era explorar les tècniques de Deep Learning i veure com es podien utilitzar per assolir l'objectiu de crear una aplicació que ens facilites dades a partir de vídeos de futbol sala.

3.3. Entrenament de models

Després d'haver fet un estudi previ en una àrea desconeguda algunes de les opcions passaven per entrenar models de detecció de elements i accions tals com xuts, passades, etc. Posteriorment utilitzar aquests models per acaba creant una aplicació que permetés la seva utilització i guardés la informació obtinguda a una base de dades.

Per començar el desenvolupament es realitzen prèviament alguns dels exemples del llibre "Deep Learning, Introducción pràctica con Keras (PRIMERA PARTE), Jordi Torres". Durant la lectura del llibre i la realització dels exemples i proves es comencen a observar diversos aspectes d'importància en el món del Deep Learning que poden dificultar i endarrerir l'objectiu final.

3.3.1. Conjunts de dades

Un dels aspectes rellevants en l'entrenament de xarxes neuronals és el conjunt de dades utilitzades per a realitzar aquesta feina.

Com s'ha vist, entrenar un model de dades amb un conjunt bastant petit pot provocar desajustos. Però no és l'únic inconvenient, la qualitat d'aquestes, també influeix en el resultat. Si les imatges utilitzades per aquest procés són totes de molt bona qualitat provocarà que el model no sigui capaç de identificar objectes o accions en imatges de menor qualitat.

També es necessari fer un pre-processament de les imatges utilitzades per adequar-les en mida, espai de colors, etiquetes.

Tenint en compte aquest aspecte el primer que es busca son conjunt de dades ja preparades per entrenar models per a detectar elements i accions pròpies de l'esport i facilitar l'objectiu. Però la realitat es que no es troben masses.

3.3.2. Computació

La xarxes neuronals artificials requereixen una gran quantitat de càlculs, per això es molt important disposar d'eines suficientment potents per realitzar les tasques d'entrenament. Actualment aquestes són possibles gràcies als avenços dels últims anys. El creixement exponencial de la capacitat de computació a sigut un dels factors claus juntament amb la aparició de les GPU que acceleren els càlculs sobre matrius numèriques i els sistemes distribuïts utilitzant diverses maquines amb múltiples GPU connectades per una xarxa.

L'ordinador del que es disposa conte una GPU però és incompatible amb la versió de la llibreria tensorflow.

La necessitat d'una potència de la que no es disposa juntament amb la problemàtica de les dades fa desviar-se del plantejament inicial buscant altres alternatives per aconseguir desenvolupar el projecte.

3.4. Models pre-entrenats

Com a solució es troba la opció d'utilitzar models pre-entrenats, que es basa en utilitzar els pesos i la arquitectura de la xarxa guardades després d'haver-lo entrenat. D'aquesta forma, al utilitzar la xarxa en un futur no farà falta tornar-la a entrenar.

Es soluciona, per tant, la problemàtica de la necessitat d'una gran quantitat de dades processades, els requeriments de hardware i es redueix el temps que es dedicaria a entrenar un model.

En aquest punt es parteix de la idea d'agafar aquest elements per al desenvolupament de la aplicació, amb la intenció de començar fent la extracció de dades més simples i un cop implementades anar desenvolupant noves funcionalitats.

3.5. Estimació de la velocitat

Inspirat en un article on es parla sobre un radar basat en tècniques d'intel·ligència artificial, que permet detectar si sobrepassa el límit de velocitat, es decideix crear una primera funcionalitat que permeti calcular la velocitat dels jugadors.

Aquesta eina esta desenvolupada pensant en ser utilitzada durant un entrenament per calcular les velocitats dels jugadors. Es basa en el concepte de calcular la velocitat seguint la seva formula matemàtica:

$$velocitat = distancia \ / \ temps$$

Per tant, per calcular-la serà necessari tenir aquests dos elements. El primer s'haurà de mesurar i passar com a paràmetre en el moment en que es vagi a utilitzar l'aplicació, col·locant la càmera, observant quina distancia recorrerà el jugador al passar per l'àrea visible d'aquesta.

S'utilitza un primer model pre-entrenat (MobileNetSSD) per a la detecció del jugador, s'estableixen dos punts A i B de la imatge (píxels) i es calcula la distancia real per píxel. Amb la detecció, sumada a un seguiment d'aquesta per identificar els diferents jugadors, es registre les posicions exactes i els temps en que es superen les marques. Finalment es mostra la velocitat calculada amb metres per segons i kilòmetres per hora. Cal remarcar que la detecció es realitza cada cert nombre de frames, dada que es passa com a paràmetre.

Per comprovar el calibratge de l'aplicació es realitzen proves amb un cotxe ja que es disposa del indicador de velocitat per comprovar el bon funcionament.

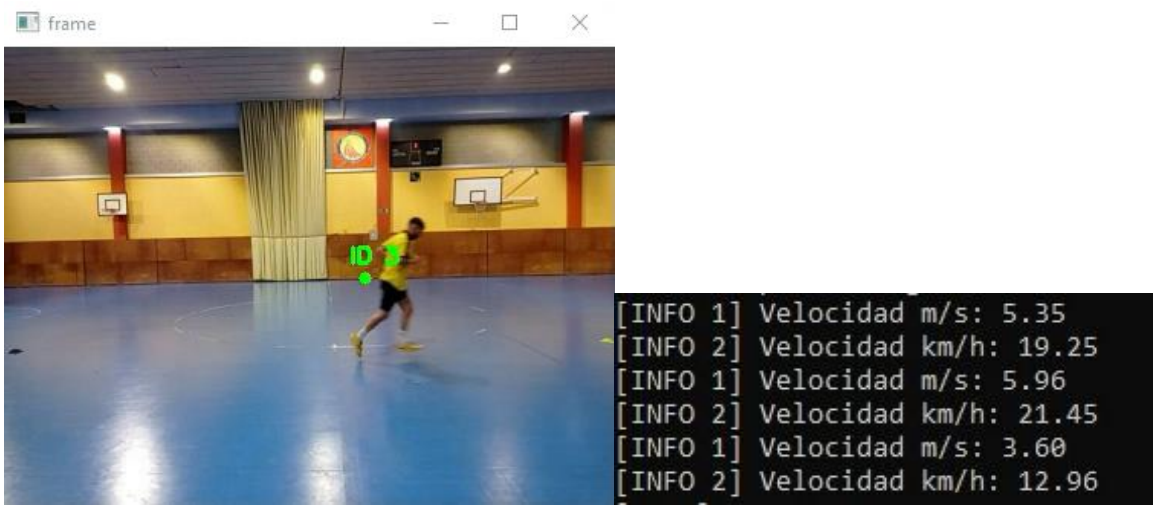


Fig. 11 – Realització de proves de l'aplicació per calcular la velocitat d'un jugador

3.5.1. Estimació de la velocitat d'una pilota

Es pensa també en reutilitzar part d'aquest codi per realitzar la mateixa funció per estimar la velocitat de la pilota en la realització d'un xut o passada.

En aquest cas es busca un altre model que permeti detectar la pilota amb uns bons resultats. Cercant se'n troba un, yolov3. En aquest cas no s'aconsegueix l'objectiu ja que

la detecció no es tant acurada com per detectar-la en la majoria de frames per a la potencia a la que es fan el xuts.

Aquest model també permet detectar persones, amb la qual cosa s'aprofita per comparar-lo amb l'anterior. Però tampoc s'arriben a obtenir millors resultats.

3.6. Comptador de temps que es situa la pilota en cada camp

El següent objectiu que es busca és recollir les dades de temps que es situa la pilota en cada camp, una estadística útil per saber si s'ha estat atacant més o menys durant un període de temps. També indica el perill estant més a prop de la porteria pròpia o rival.

S'aprofita el model pre-entrenat utilitzat, es calcula el punt mig (tenint en compte que es farà coincidir aquest amb el mig camp) i es detecta i registre la pilota per tal de calcular en quin moment canvia d'una banda a l'altre d'aquest punt. És a dir, es fixa en l'eix de les x per saber la detecció de la pilota es troba a l'esquerra o la dreta d'aquest i amb això identificar en quin instant de temps a canvia d'un a l'altre. Aquest temps es suma a un comptador i cada vegada que s'ha fet un canvi de camp es mostra per pantalla quant de temps ha estat en cada un.

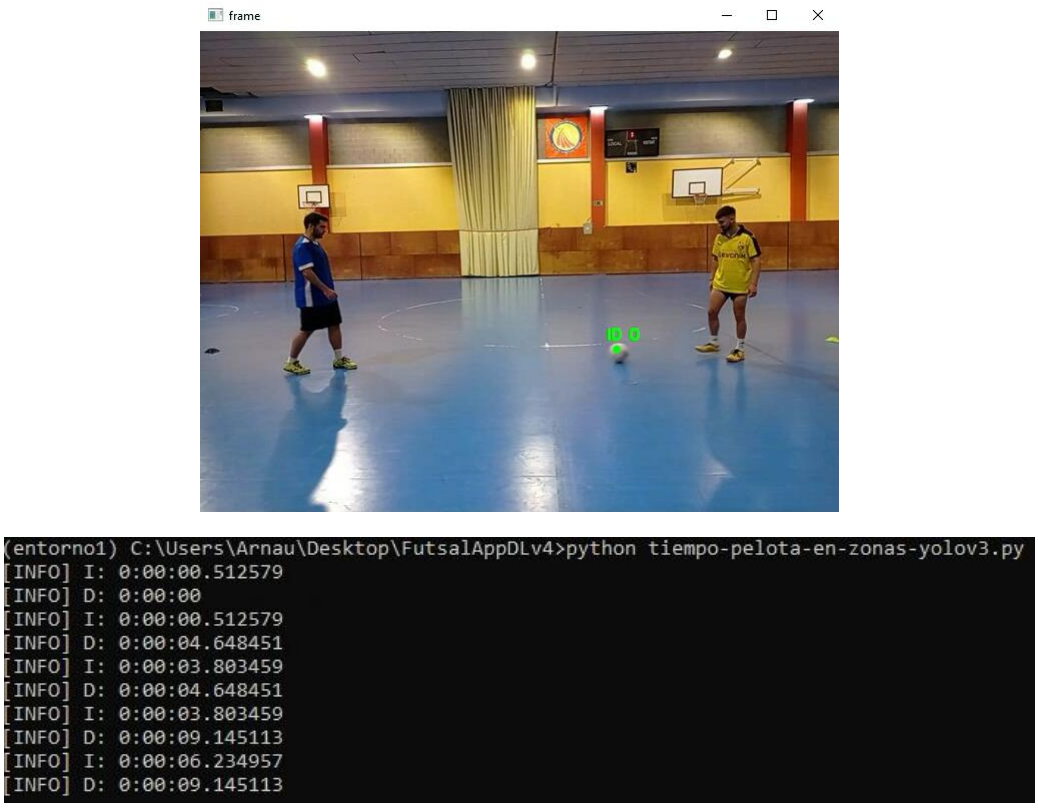


Fig. 12 – Realització de proves de l'aplicació per calcular el temps de la pilota en cada meitat de camp

S'ha de remarcar que totes les funcionalitats han d'utilitzar-se en temps real ja que el temps es un factor importat per els càlculs. Si el processat d'un vídeo ja gravat es més ràpid els registres de temps en cada punt no seran reals.

4. Resultats

En aquest capítol s'analitzaran els resultats obtinguts en desenvolupament de la aplicació.

Analitzant els mòduls / aplicacions per separat comencem per la estimació de la velocitat en jugadors. El primer que observarem es si s'assoleix la funcionalitat, en aquets cas podem dir que s'ha aconseguit mesurar correctament les dades. Tot i això es detecten petits errors o possibles millores.

En el cas del model de yolov3 els requadres delimitadors de les deteccions en alguns frames no s'acaben d'ajustar del tot fent variar el centre i provocant petites variacions en els càlculs. També hi havia algun exemple en el que no s'aconseguia estimar la veolictat, com pot ser en velocitats altes. Segurament amb un model més centrat en la detecció de persones en moviment a altes velocitats es podria haver obtingut un resultat millor.

En el model MobileNetSSD es veuen millores, els quadres estan més ajustats, les velocitats assolides són millors. Cal remarcar que la mida de les dades d'entrada també influeix en aquest aspecte ja que amb menor quantitat de dades es requereixen menys càlculs.

En el cas d'estimar la velocitat d'una pilota els resultats han estat bastant dolents fins al punt de no aconseguir una funcionalitat correcte.

En la funcionalitat de calcular el temps que la pilota està en cada meitat del camp, s'obtenen bons resultats. Els petits errors com poden ser perdre alguna detecció concreta no afecten tant al resultat final. Si la detecció s'ha perdut en un moment on no es canvia de camp no influirà en res, només en els cassos on es canvia de camp però de totes maneres la diferencia serà de pocs segons. Tenint en compte que l'important es detectar si s'ha estat en un camp molt més que en un altre, petites variacions no afectarien en el resultat de l'anàlisi.

Per altre banda si analitzem els objectius proposats a l'inici del projecte s'observa que s'assoleixen la majoria:

- S'ha realitzat un estudi sobre les tècniques de Deep Learning i observat les possibilitats d'aplicar-les en el món del futbol sala.
- Per desenvolupar el prototip s'ha utilitzat un llenguatge que no havia utilitzat prèviament. Ampliant els coneixements en programació.
- S'ha aconseguit aplicar algunes tècniques estudiades creant unes aplicacions.

Però tot i així hi ha hagut alguns objectius que s'han quedat en el camí:

- La utilització de bases de dades per emmagatzemar la informació extreta no ha estat possible degut a que el temps era limitat i hi havia molts conceptes nous a aprendre.
- També m'hagués agradat poder construir una aplicació més visual amb una Interface gràfica per fer-la més agradable.

5. Pressupost

Per el desenvolupament del projecte s'han utilitzat algunes eines gratuïtes:

- Python
- Sublime Text
- DroidCampApp

Per altre banda tenim els costos d'alguns components necessaris per el desenvolupament del software i realització de proves :

Component	Cost
Ordinador	500 eur
Smartphone	100 eur

Taula 4 – Costos dels components

Els costos dels components s'han calculat aproximadament fent el càlcul de tenint en compte que s'han utilitzat un any i la seva vida útil. Restant del total el que seria el valor residual.

També s'ha de tenir en compte les hores dedicades:

Rol	Cost
Enginyer junior	12 eur/h x 540 h = 6480 eur
Supervisor	25 eur/h x 21 h = 525

Taula 5 – Costos de les hores

El cost total obtingut és de 7605 euros.

6. Conclusions i desenvolupament futur

Les principals conclusions extretes i aspectes a tenir en compte per a la realització d'un projecte d'aquest àmbit són:

Els alts requeriments necessaris per poder entrenar i utilitzar un model, un bon conjunt de dades ben preparades es molt important per obtenir un model precís. També és necessita tenir una màquina potent per el volum elevat de càlculs necessaris tant per l'entrenament com per el processat d'imatges.

També que tot i tenir aquests requeriments les possibilitats d'aplicar les tècniques Deep Learning en el futbol sala són moltes.

En quant al treball futur, amb la informació recaptada es pot continuar el projecte augmentant les seves funcionalitat.

Un primer camí seria assolir els objectius que s'han pogut aconseguir en la realització d'aquest treball. També trobem la possibilitat recol·lectar dades i processar-les per tal d'entrenar models propis més especialitzats i canviar els utilitzats per aquests.

Un altre objectiu seria anar ampliant les funcionalitats de l'aplicació, com per exemple, fer un seguiment dels jugadors (temps i posició), recollir dades accions importants com numero de xuts i passades, o de la possessió de cada equip entre d'altres.

Bibliografia:

Welcome to Python.org, from https://www.python.org/

Keras: the Python deep learning API, from https://keras.io/

Home - OpenCV, from https://opencv.org/

Sublime Text - Text Editing, Done Righ, from https://www.sublimetext.com/

Cómo usar la cámara de tu móvil Android como webcam para tu PC, from https://www.xatakandroid.com/tutoriales/como-usar-camara-tu-movil-android-como-webcam-para-tu-pc

Windows | Dev47Apps, from https://www.dev47apps.com/droidcam/windows/

Deep Learning – Introducción práctica con Keras - Jordi TORRES.AI, from https://torres.ai/deep-learning-inteligencia-artificial-keras

Deep Learning, Introducción práctica con Keras (SEGUNDA PARTE) - Jordi TORRES.AI, from https://torres.ai/deep-learning-inteligencia-artificial-keras-2a-parte/

Deep Learning con redes pre-entrenadas en ImageNet | by Daniel Lerch | Neuron4 | Medium, from https://medium.com/neuron4/redes-pre-entrenadas-en-imagenet-30d858c37b1f

What is Overfitting? | IBM, from https://www.ibm.com/cloud/learn/overfitting

¿Qué es la Función de Activación?, from https://luca-d3.com/es/data-speaks/diccionario-tecnologico/funcion-activacion

What is Deep Learning? | IBM, from https://www.ibm.com/cloud/learn/deep-learning

¿Qué es la inteligencia artificial? | Microsoft Azure, from https://azure.microsoft.com/es-es/overview/what-is-artificial-intelligence/#types

¿Qué es la inteligencia artificial (IA)? - España | IBM, from https://www.ibm.com/es-es/cloud/learn/what-is-artificial-intelligence

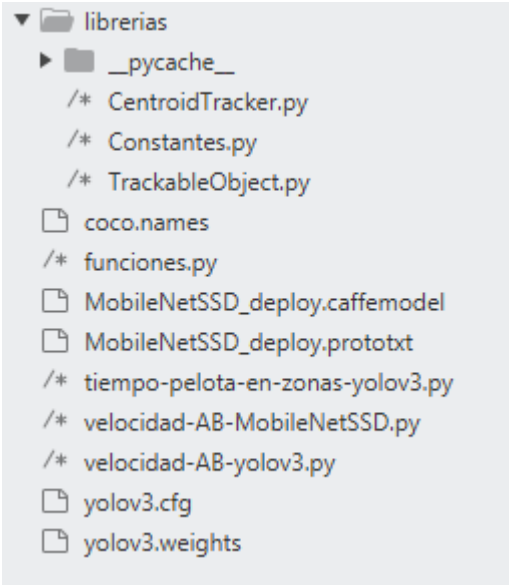
Start Here with Computer Vision, Deep Learning, and OpenCV - PyImageSearch, from https://www.pyimagesearch.com/start-here/

Social_distancing_using_caffe_model/models at main · arshilb4u/Social_distancing_using_caffe_model · GitHub, from https://github.com/arshilb4u/Social_distancing_using_caffe_model/tree/main/models

YOLO: Real-Time Object Detection, from https://pjreddie.com/darknet/yolo/

Annexes

Estructura



Codi

Constantes

```python
import cv2
import numpy as np

class cts_y:

    DISTANCIA_REAL = 7.67
    CONFIANZA_MIN = 0.5

    FRAMES_DESAPARECIDO = 10
    DISTANCIA_MISMO_OBJETO = 175
    FRAMES_SEGUIMIENTO = 4
    CLASE_DETECTAR = 32 #0-persona, 32-pelota

    ANCHO_ENTRADA = 416
    ALTO_ENTRADA = 416

    MOSTRAR_VIDEO = True

    RUTA_VIDEO_ENTRADA = 0
    RUTA_FICHERO_SALIDA = "salida.csv"

    CFG_MODELO = "yolov3.cfg"
    PESOS_MODELO = "yolov3.weights"
    FICHERO_CLASES = "coco.names"

class cts_mn:

    DISTANCIA_REAL = 7.67
    CONFIANZA = 0.4

    FRAMES_DESAPARECIDO = 10
    DISTANCIA_MISMO_OBJETO = 175
    FRAMES_SEGUIMIENTO = 4
    CLASES = ["background", "aeroplane", "bicycle", "bird", "boat", "bottle", "bus", "car", "cat", "chair", "cow", "diningtable", "dog", "horse", "motorbike", "person",
    CLASE_DETECTAR = "person"

    ANCHO_FRAME = 400
    PUNTO_CONTROL = {"A":120, "B":280}

    MOSTRAR_VIDEO = True  #Mostrar video con la deteccion

    RUTA_FICHERO = "output"
    NOMBRE_CSV = "log.csv"

    RUTA_MODELO = "MobileNetSSD_deploy.caffemodel"
    RUTA_PROTOTXT = "MobileNetSSD_deploy.prototxt"
```

CentroidTracker

```python
from scipy.spatial import distance as dist
from collections import OrderedDict
import numpy as np

class CentroidTracker():
    def __init__(self, framesDesaparicion=50, distanciaMaxima=175):
        self.proximoId = 0
        self.objetos = OrderedDict()
        self.desapariciones = OrderedDict()
        self.framesDesaparicion = framesDesaparicion
        self.distanciaMaxima = distanciaMaxima

    def registrar(self, centroide):
        self.objetos[self.proximoId] = centroide
        self.desapariciones[self.proximoId] = 0
        self.proximoId += 1

    def eliminar(self, idObjeto):
        del self.objetos[idObjeto]
        del self.desapariciones[idObjeto]

    def actualizar(self, cuadros):
        # Si no hay cuadros (detecciones), sumar desapariciones de los registrados
        if len(cuadros) == 0:
            for idObjeto in list(self.desapariciones.keys()):
                self.desapariciones[idObjeto] += 1
                if self.desapariciones[idObjeto] > self.framesDesaparicion:
                    self.eliminar(idObjeto)
            return self.objetos

        # Calcular el centro de cada cuadro delimitador y guardarlos en un array
        centroidesEntrada = np.zeros((len(cuadros),2),dtype="int")
        for (i, (inicioX, inicioY, finalX, finalY)) in enumerate(cuadros):
            cX = int((inicioX + finalX) / 2.0)
            cY = int((inicioY + finalY) / 2.0)
            centroidesEntrada[i] = (cX,cY)

        # Si no hay objetos registrados registrarlos todos
        if len(self.objetos) == 0:
            for i in range(0,len(centroidesEntrada)):
                self.registrar(centroidesEntrada[i])
        else:
            idsObjetos = list(self.objetos.keys())
            centroidesObjetos = list(self.objetos.values())

            # Calcular todas las distancias entre registrados y encontrados
            D = dist.cdist(np.array(centroidesObjetos),centroidesEntrada)
```

```
            # Ordenamos por distancias minimas de filas y columnas
            filas = D.min(axis=1).argsort()
            columnas = D.argmin(axis=1)[filas]

            filasUsadas = set()
            columnasUsadas = set()
            for (fila, columna) in zip(filas, columnas):
                if fila in filasUsadas or columna in columnasUsadas:
                    continue
                idObjeto = idsObjetos[fila]
                self.objetos[idObjeto] = centroidesEntrada[columna]
                self. desapariciones[idObjeto] = 0

                filasUsadas.add(fila)
                columnasUsadas.add(columna)

            filasNoUsadas = set(range(0,D.shape[0])).difference(filasUsadas)
            columnasNoUsadas = set(range(0,D.shape[1])).difference(columnasUsadas)
            if D.shape[0] >= D.shape[1]:
                for fila in filasNoUsadas:
                    idobjeto = idsObjetos[fila]
                    self.desapariciones[idobjeto] += 1

                    if self.desapariciones[idObjeto] > self.framesDesaparicion:
                        self.eliminar(idObjeto)
            else:
                for columna in columnasNoUsadas:
                    self.registrar(centroidesEntrada[columna])

        return self.objetos
```

TrackableObject

```
import numpy as np
from datetime import datetime, timedelta

class TrackableObject:
    def __init__(self, idObjeto, centroide):
        self.idObjeto = idObjeto
        self.centroides = [centroide]
        self.tiempos = {"A": 0, "B": 0, "C": 0, "D": 0}
        self.posicion = {"A": None, "B": None, "C": None, "D": None}
        self.ultimoPunto = False
        self.velocidadMPS = None
        self.velocidadKMPH = None
        self.estimado = False
        self.registrado = False
        self.direccion = None

    def calcular_velocidad(self, velocidadesEstimadas):
        self.velocidadMPS = np.average(velocidadesEstimadas)
        self.velocidadKMPH = self.velocidadMPS * 3.6

    def mpsAkmph(self):
        self.velocidadKMPH = self.velocidadMPS * 3.6

class TrackableObjectZonas:
    def __init__(self, idObjeto, centroide, tiempoActual):
        self.idObjeto = idObjeto
        self.centroides = [centroide]
        self.tiempo = {"I": timedelta(days=0, seconds=0, microseconds=0, milliseconds=0, minutes=0, hours=0, weeks=0),
        "D": timedelta(days=0, seconds=0, microseconds=0, milliseconds=0, minutes=0, hours=0, weeks=0),
        "C": timedelta(days=0, seconds=0, microseconds=0, milliseconds=0, minutes=0, hours=0, weeks=0)}
        self.ultimaPosicion = "C"
        self.tiempoUltimoCambio = tiempoActual
```

```python
class TrackableObjectVelocidadAB:
    #CONSTRUCTOR
    def __init__(self, idObjeto, centroide):
        self.idObjeto = idObjeto
        self.centroides = [centroide]
        self.tiempos = {"A": 0, "B": 0}
        self.posicion = {"A": None, "B": None}
        self.ultimoPunto = False
        self.velocidadMPS = None
        self.velocidadKMPH = None
        self.estimado = False
        self.registrado = False
        self.direccion = None

    def calcular_velocidad(self, velocidadesEstimadas):
        self.velocidadMPS = np.average(velocidadesEstimadas)
        self.velocidadKMPH = self.velocidadMPS * 3.6

    def mpsAkmph(self):
        self.velocidadKMPH = self.velocidadMPS * 3.6

class TrackableObjectPelota:
    def __init__(self, idPelota, centro, tUltimoCambio):
        self.idPelota = idPelota
        self.centro = [centro]
        self.tiempoEn = {"I": timedelta(days=0), "D": timedelta(days=0)}
        self.ultimaPosicion = "None"
        self.tUltimoCambio = tUltimoCambio

    def actualizarPosicion(self, centroImagen, instanteActual, centro):
        if self.ultimaPosicion != "D" and centro > centroImagen:
            tiempoSumado = instanteActual - self.tUltimoCambio
            self.tiempoEn["I"] = self.tiempoEn["I"] + tiempoSumado
            self.ultimaPosicion = "D"
            self.tUltimoCambio = instanteActual
            print("[INFO] I: {}".format(self.tiempoEn["I"]))
            print("[INFO] D: {}".format(self.tiempoEn["D"]))
        elif self.ultimaPosicion != "I" and centro < centroImagen:
            tiempoSumado = instanteActual - self.tUltimoCambio
            self.tiempoEn["D"] = self.tiempoEn["D"] + tiempoSumado
            self.ultimaPosicion = "I"
            self.tUltimoCambio = instanteActual
            print("[INFO] I: {}".format(self.tiempoEn["I"]))
            print("[INFO] D: {}".format(self.tiempoEn["D"]))
```

Funciones

```python
import cv2
import numpy as np
from librerias.Constantes import cts_y
import dlib

class funciones:

    confThreshold = 0.5
    nmsThreshold = 0.4
    inpWidth = 416
    inpHeight = 416

    def getOutputsNames(net):
        layersNames = net.getLayerNames()
        return [layersNames[i[0] - 1] for i in net.getUnconnectedOutLayers()]

    def ppVelocidad(frame, detecciones, boxesFinales, trackers, rgb):
        nmsThreshold = 0.4
        altoFrame = frame.shape[0]
        frameWidth = frame.shape[1]
        classIds = []
        confidences = []
        boxes = []
        for out in detecciones:
            for detection in out:
                scores = detection[5:]
                classId = np.argmax(scores)
                confidence = scores[classId]
                if classId != cts_y.CLASE_DETECTAR:
                    continue
                if confidence > cts_y.CONFIANZA_MIN:
                    center_x = int(detection[0] * frameWidth)
                    center_y = int(detection[1] * altoFrame)
                    width = int(detection[2] * frameWidth)
                    height = int(detection[3] * altoFrame)
                    left = int(center_x - width / 2)
                    top = int(center_y - height / 2)
                    classIds.append(classId)
                    confidences.append(float(confidence))
                    boxes.append([left, top, width, height])
```

```
        indices = cv2.dnn.NMSBoxes(boxes, confidences, cts_y.CONFIANZA_MIN, nmsThreshold)
        for i in indices:
            i = i[0]
            box = boxes[i]
            left = box[0]
            top = box[1]
            width = box[2]
            height = box[3]

            classId = classIds[i]
            conf = confidences[i]
            right = left + width
            bottom = top + height

            boxesFinales.append([left, top, right, bottom])
            #cv2.rectangle(frame, (left, top), (right, bottom), (255, 178, 50), 3)

            tracker = dlib.correlation_tracker()
            rect = dlib.rectangle(left, top, left+width, top+height)
            tracker.start_track(rgb, rect)
            trackers.append(tracker)
```

Tiempo-pelota-en-zonas-yolov3

```
from funciones import funciones
from librerias.CentroidTracker import CentroidTracker
from librerias.TrackableObject import TrackableObjectPelota
from librerias.Constantes import cts_y
from imutils.video import VideoStream
from imutils.io import TempFile
from imutils.video import FPS
from datetime import datetime, timedelta
from threading import Thread
import numpy as np
import argparse
import imutils
import dlib
import time
import cv2
import os

altoFrame = None
anchoFrame = None
metrosPorPixel = None
trackers = []
trackableObjects = {}
framesTotales = 0
ct = CentroidTracker(framesDesaparicion=cts_y.FRAMES_DESAPARECIDO, distanciaMaxima=cts_y.DISTANCIA_MISMO_OBJETO)

modelo = cv2.dnn.readNetFromDarknet(cts_y.CFG_MODELO, cts_y.PESOS_MODELO)
video = cv2.VideoCapture(cts_y.RUTA_VIDEO_ENTRADA)

while cv2.waitKey(1) & 0xFF != ord("q"):
    boxesFinales=[]
    blnFrame, frame = video.read()
    rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    instanteActual = datetime.now()

    if anchoFrame is None or altoFrame is None:
        (altoFrame, anchoFrame) = frame.shape[:2]
        centroImagen = anchoFrame/2
    if framesTotales % cts_y.FRAMES_SEGUIMIENTO == 0:
        blob = cv2.dnn.blobFromImage(frame, 1/255, (cts_y.ANCHO_ENTRADA, cts_y.ALTO_ENTRADA), [0,0,0], 1, crop=False)
        modelo.setInput(blob)

        namesOutputs = funciones.getOutputsNames(modelo)
        detecciones = modelo.forward(namesOutputs)
        funciones.ppVelocidad(frame, detecciones, boxesFinales, trackers, rgb)
        t, _ = modelo.getPerfProfile()
```

```
    else:
        for tracker in trackers:
            tracker.update(rgb)
            pos = tracker.get_position()
            inicioX = int(pos.left())
            inicioY = int(pos.top())
            finalX = int(pos.right())
            finalY = int(pos.bottom())
            boxesFinales.append((inicioX, inicioY, finalX, finalY))

    objects = ct.actualizar(boxesFinales)
    for (idObjeto, centroid) in objects.items():
        to = trackableObjects.get(idObjeto, None)
        if to is None:
            to = TrackableObjectPelota(idObjeto, centroid, instanteActual)
        else:
            to.actualizarPosicion(centroImagen, instanteActual, centroid[0])

        trackableObjects[idObjeto] = to
        if cts_y.MOSTRAR_VIDEO:
            text = "ID {}".format(idObjeto)
            cv2.putText(frame, text, (centroid[0] - 10, centroid[1] - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
            cv2.circle(frame, (centroid[0], centroid[1]), 4,(0, 255, 0), -1)

    if cts_y.MOSTRAR_VIDEO:
        cv2.imshow("frame", frame)

video.release()
cv2.destroyAllWindows()
```

velocidad-AB-MobileNetSSD

```
    if framesTotales % cts_mn.FRAMES_SEGUIMIENTO == 0:
        trackers = []
        blob = cv2.dnn.blobFromImage(frame, size=(300, 300), ddepth=cv2.CV_8U)
        modelo.setInput(blob, scalefactor=1.0/127.5, mean=[127.5, 127.5, 127.5])
        detecciones = modelo.forward()

        for i in np.arange(0, detecciones.shape[2]):
            confianza = detecciones[0, 0, i, 2]
            if confianza > cts_mn.CONFIANZA:
                idx = int(detecciones[0, 0, i, 1])
                if cts_mn.CLASES[idx] != cts_mn.CLASE_DETECTAR:
                    continue
                box = detecciones[0, 0, i, 3:7] * np.array([anchoFrame, altoFrame, anchoFrame, altoFrame])
                (inicioX, inicioY, finalX, finalY) = box.astype("int")
                tracker = dlib.correlation_tracker()
                rect = dlib.rectangle(inicioX, inicioY, finalX, finalY)
                tracker.start_track(rgb, rect)
                trackers.append(tracker)
    else:
        for tracker in trackers:
            tracker.update(rgb)
            pos = tracker.get_position()
            inicioX = int(pos.left())
            inicioY = int(pos.top())
            finalX = int(pos.right())
            finalY = int(pos.bottom())
            rects.append((inicioX, inicioY, finalX, finalY))

    objects = ct.actualizar(rects)

    for (idObjeto, centroid) in objects.items():
        to = trackableObjects.get(idObjeto, None)
        if to is None:
            to = TrackableObject(idObjeto, centroid)
        elif not to.estimado:
            if to.direccion is None:
                y = [c[0] for c in to.centroides]
                direction = centroid[0] - np.mean(y)
                to.direccion = direction
```

```
from librerias.CentroidTracker import CentroidTracker
from librerias.TrackableObject import TrackableObject
from librerias.Constantes import cts_mn
from imutils.video import VideoStream
from imutils.io import TempFile
from imutils.video import FPS
from datetime import datetime
from threading import Thread
import numpy as np
import argparse
import imutils
import dlib
import time
import cv2
import os

altoFrame = None
anchoFrame = None
ct = CentroidTracker(framesDesaparicion=cts_mn.FRAMES_DESAPARECIDO, distanciaMaxima=cts_mn.DISTANCIA_MISMO_OBJETO)
trackers = []
trackableObjects = {}
framesTotales = 0

modelo = cv2.dnn.readNetFromCaffe(cts_mn.RUTA_PROTOTXT, cts_mn.RUTA_MODELO) #Cargar modelo preentrenado
modelo.setPreferableTarget(cv2.dnn.DNN_BACKEND_INFERENCE_ENGINE)

video = VideoStream(src=0).start()
time.sleep(2.0)
fps = FPS().start()

while True:
    frame = video.read()
    instanteActual = datetime.now()

    if frame is None:
        break

    frame = imutils.resize(frame, width=cts_mn.ANCHO_FRAME)
    rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

    if anchoFrame is None or altoFrame is None:
        (altoFrame, anchoFrame) = frame.shape[:2]
        metrosPorPixel = cts_mn.DISTANCIA_REAL / anchoFrame

    rects = []
```

```
            if to.direccion > 0:
                if to.tiempos["A"] == 0 and centroid[0] > cts_mn.PUNTO_CONTROL["A"]:
                        to.tiempos["A"] = instanteActual
                        to.posicion["A"] = centroid[0]
                elif to.tiempos["B"] == 0 and centroid[0] > cts_mn.PUNTO_CONTROL["B"]:
                        to.tiempos["B"] = instanteActual
                        to.posicion["B"] = centroid[0]
                        to.ultimoPunto = True
            elif to.direccion < 0:
                if to.tiempos["B"] == 0 and centroid[0] < cts_mn.PUNTO_CONTROL["B"]:
                        to.tiempos["B"] = instanteActual
                        to.posicion["B"] = centroid[0]
                elif to.tiempos["A"] == 0 and centroid[0] < cts_mn.PUNTO_CONTROL["A"]:
                        to.tiempos["A"] = instanteActual
                        to.posicion["A"] = centroid[0]
                        to.ultimoPunto = True

            if to.ultimoPunto:
                distanciaPixeles = abs(to.posicion["B"] - to.posicion["A"])
                if distanciaPixeles == 0:
                    continue
                t = to.tiempos["B"] - to.tiempos["A"]
                segundos = abs(t.total_seconds())
                distanciaMetros = distanciaPixeles * metrosPorPixel
                to.velocidadMPS = distanciaMetros / segundos
                to.mpsAkmph()
                to.estimado = True
                print("[INFO 1] Velocidad m/s: {:.2f}".format(to.velocidadMPS))
                print("[INFO 2] Velocidad km/h: {:.2f}".format(to.velocidadKMPH))

        trackableObjects[idObjeto] = to
        if cts_mn.MOSTRAR_VIDEO:
            text = "ID {}".format(idObjeto)
            cv2.putText(frame, text, (centroid[0] - 10, centroid[1] - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
            cv2.circle(frame, (centroid[0], centroid[1]), 4,(0, 255, 0), -1)

    if cts_mn.MOSTRAR_VIDEO:
        cv2.imshow("frame", frame)
        key = cv2.waitKey(1) & 0xFF
        if key == ord("q"):
            break
```

```
    framesTotales += 1
    fps.update()

fps.stop()
print("[INFO] FPS: {:.2f}".format(fps.fps()))

cv2.destroyAllWindows()
video.stop()
```

velocidad-AB-yolov3

```
from funciones import funciones
from librerias.CentroidTracker import CentroidTracker
from librerias.TrackableObject import TrackableObjectVelocidadAB
from librerias.Constantes import cts_y
from imutils.video import VideoStream
from imutils.io import TempFile
from imutils.video import FPS
from datetime import datetime
from threading import Thread
import numpy as np
import argparse
import imutils
import dlib
import time
import cv2
import os

altoFrame = None
anchoFrame = None
metrosPorPixel = None
trackers = []
trackableObjects = {}
framesTotales = 0
puntoControl = {"A": None, "B": None}
ct = CentroidTracker(framesDesaparicion=cts_y.FRAMES_DESAPARECIDO, distanciaMaxima=cts_y.DISTANCIA_MISMO_OBJETO)

modelo = cv2.dnn.readNetFromDarknet(cts_y.CFG_MODELO, cts_y.PESOS_MODELO)
video = cv2.VideoCapture(cts_y.RUTA_VIDEO_ENTRADA)

while cv2.waitKey(1) & 0xFF != ord("q"):
    boxesFinales=[]
    bln, frame = video.read()
    rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    instanteActual = datetime.now()

    if anchoFrame is None or altoFrame is None:
        (altoFrame, anchoFrame) = frame.shape[:2]
        metrosPorPixel = cts_y.DISTANCIA_REAL / anchoFrame
    elif puntoControl["A"] is None or puntoControl["B"] is None:
        puntoControl["A"] = anchoFrame * 0.3
        puntoControl["B"] = anchoFrame - (anchoFrame * 0.3)

    if framesTotales % cts_y.FRAMES_SEGUIMIENTO == 0:
        trackers = []
        blob = cv2.dnn.blobFromImage(frame, 1/255, (cts_y.ANCHO_ENTRADA, cts_y.ALTO_ENTRADA), [0,0,0], 1, crop=False)
        modelo.setInput(blob)
```

```
            namesOutputs = funciones.getOutputsNames(modelo)
            detecciones = modelo.forward(namesOutputs)
            funciones.ppVelocidad(frame, detecciones, boxesFinales, trackers, rgb)
            t, _ = modelo.getPerfProfile()
        else:
            for tracker in trackers:
                tracker.update(rgb)
                pos = tracker.get_position()
                inicioX = int(pos.left())
                inicioY = int(pos.top())
                finalX = int(pos.right())
                finalY = int(pos.bottom())
                boxesFinales.append((inicioX, inicioY, finalX, finalY))

        objects = ct.actualizar(boxesFinales)
        for (idObjeto, centroid) in objects.items():
            to = trackableObjects.get(idObjeto, None)
            if to is None:
                to = TrackableObjectVelocidadAB(idObjeto, centroid)
            elif not to.estimado:
                if to.direccion is None:
                    y = [c[0] for c in to.centroides]
                    direction = centroid[0] - np.mean(y)
                    to.direccion = direction

                if to.direccion > 0:
                    if to.tiempos["A"] == 0 and centroid[0] > puntoControl["A"]:
                            to.tiempos["A"] = instanteActual
                            to.posicion["A"] = centroid[0]
                    elif to.tiempos["B"] == 0 and centroid[0] > puntoControl["B"]:
                            to.tiempos["B"] = instanteActual
                            to.posicion["B"] = centroid[0]
                            to.ultimoPunto = True
                elif to.direccion < 0:
                    if to.tiempos["B"] == 0 and centroid[0] < puntoControl["B"]:
                            to.tiempos["B"] = instanteActual
                            to.posicion["B"] = centroid[0]
                    elif to.tiempos["A"] == 0 and centroid[0] < puntoControl["A"]:
                            to.tiempos["A"] = instanteActual
                            to.posicion["A"] = centroid[0]
                            to.ultimoPunto = True
```

```
            if to.ultimoPunto:
                distanciaPixeles = abs(to.posicion["B"] - to.posicion["A"])
                if distanciaPixeles == 0:
                    continue
                t = to.tiempos["B"] - to.tiempos["A"]
                segundos = abs(t.total_seconds())
                distanciaMetros = distanciaPixeles * metrosPorPixel
                to.velocidadMPS = distanciaMetros / segundos
                to.mpsAkmph()
                print("[INFO 1] Velocidad m/s: {:.2f}".format(to.velocidadMPS))
                print("[INFO 2] Velocidad km/h: {:.2f}".format(to.velocidadKMPH))
                to.estimado = True

            trackableObjects[idObjeto] = to
            if cts_y.MOSTRAR_VIDEO:
                text = "ID {}".format(idObjeto)
                cv2.putText(frame, text, (centroid[0] - 10, centroid[1] - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
                cv2.circle(frame, (centroid[0], centroid[1]), 4,(0, 255, 0), -1)

        if cts_y.MOSTRAR_VIDEO:
            cv2.imshow("frame", frame)

    video.release()
    cv2.destroyAllWindows()
```