Universitat de Girona
Escola Politècnica Superior

Projecte fi de grau

Estudi: Grau en Enginyeria Informàtica

Títol: Desenvolupament d'una xarxa neuronal profunda per la ramaderia de precisió

Document: Memòria

Alumne: Virginia Ramón Ferrer

Tutor: Rafael Garcia Campos
Departament: Arquitectura i Tecnologia de Computadors
Àrea: ATC

Convocatòria (mes/any): Febrer 2022

Universitat de Girona
Escola Politècnica Superior

PROJECTE FI DE GRAU

Desenvolupament d'una xarxa neuronal profunda per la ramaderia de precisió

Autor:
Virginia RAMÓN FERRER

Febrer 2022

Grau en Enginyeria Informàtica

Tutor:
Rafael GARCIA CAMPOS

Agraïments

Per començar vull agrair a la meva familia, per donar-me tot el seu suport des de sempre en tot el que faig i demostrar-me l'orgullosos que estan de mi.

Al meu tutor Rafael Garcia, per mostrar-me una part de la informàtica que no havia descobert i confiar en mi des del primer moment, mostrant-me que puc arribar més lluny del que jo hauria pensat mai.

A VICOROB, i en especial als membres del CIRS, per ajudar-me sempre que ho he necessitat i rebre'm amb els braços oberts.

Als professors, companys i amics que m'han acompanyat al llarg de la carrera i han aportat el seu gra de sorra a que avui sigui qui soc.

De cor, gràcies.

Índex

Capítol 1

Introducció

L'arribada de les tecnologies de la informació i la comunicació a la ramaderia permet l'adquisició de dades directament a les granges, i el seu processament per a convertir-les en informació útil per a la presa de decisions sobre el maneig a realitzar. En aquest context, les granges actuals tendeixen a incrementar el nombre d'animals per treballador, al temps que el benestar animal és cada dia més rellevant.

La tecnologia a desenvolupar en aquest projecte permetrà supervisar un grup de porcs 24 hores al dia, 7 dies a la setmana, a partir d'una càmera zenital que cobreix la zona on es troben els animals. Els animals, en funció de la seva edat, estat fisiològic, i altres factors, dediquen una sèrie d'hores més o menys determinades a descansar, menjar, beure, interactuar.... És el que s'anomena el seu "pressupost diari del temps". Quan un animal o un grup, es desvia significativament del seu "pressupost horari" pot ser indicatiu d'un problema.

Aquest projecte es basa en el treball fet per Bergamini et al. [1], on es vol fer un anàlisi del pressupost diari del temps de cada animal per a analitzar com es comporten. En el cas d'aquest projecte, ens centrarem en la detecció d'animals i persones per a posteriorment veure com afecta la presència de persones al comportament de cada porc, és a dir, l'efecte dels factors externs al comportament dels animals.

Es partirà d'una arquitectura CNN (Convolutional Neural Network) pre-entrenada per la detecció d'objectes, la YOLOv4, i es modificaran les darreres capes d'aquesta xarxa neuronal per detectar porcs i persones. Inicialment només es disposava d'una base de dades que contenia exclusivament porcs i una altra que contenia només persones. Posteriorment s'han obtingut imatges on hi ha la presencia tant de porcs com a persones per a poder fer proves amb ambdues. Aquestes noves imatges no estaven etiquetades, per tant s'ha hagut d'extreure una petita mostra i etiquetar-la per a poder afegir-les als entrenaments i proves.

També s'ha desenvolupat un algorisme d'anàlisi d'imatge que interpreta les deteccions de la CNN, de forma que es mesura el moviment de cada animal al llarg del temps. Aquestes deteccions fetes per la CNN poden ser errònies, ja que els

porcs són animals que tendeixen a estar molt junts i la CNN pot detectar varis animals junts com a un sol porc o, si hi ha algun objecte al mig del porc i la càmera, detectar un sòl porc com a varis animals. L'algorisme utilitzat per a mesurar el moviment, l'Unscented Kalman Filter, ens permet en part contrarestar aquest fenomen gràcies a la predicció que pot fer quan no hi ha mesura.

Tot i que aquest PFG se centra en la detecció i seguiment del moviment dels porcs per qüestions de temps, també es vol entrenar la xarxa per a la detecció de persones per a deixar-la llesta per a utilitzar aquestes mesures i la detecció de presència de persones per a analitzar com afecta aquesta presència.

La implementació s'ha realitzat en Python i s'ha fet servir les llibreries Tensorflow i Keras per a la implementació de la xarxa neuronal.

Al capítol 2 s'explica la viabilitat i els requisits del sistema implementat. Seguidament, al capítol 3, s'explica tot el procés seguit al llarg del projecte. A continuació es mostra la planificació seguida, al capítol 4. Al capítol 5 s'introdueixen els conceptes previs necessaris per a entendre el funcionament del projecte. Al capítol 6 es presenta de forma més detallada l'estudi i les decisions d'aquest projecte. Seguidament, al capítol 7, es detalla l'anàlisi i el disseny del sistema, seguit de la implementació d'aquest i les proves realitzades, incloses al capítol 8. A continuació, al capítol 9, es mostren els resultats extrets de les proves realitzades. Finalment es presenten les conclusions extretes, al capítol 10, i el treball futur, al capítol 11.

CAPÍTOL 2

Viabilitat i requisits del sistema

Si es parla de viabilitat del projecte, hi ha tres possibles limitacions que impedeixin el correcte desenvolupament d'aquest: tecnològiques, temporals i econòmiques. Al llarg de l'estudi de viabilitat, també s'introduiran els requisits necessaris per a desenvolupar aquest sistema.

2.1 Viabilitat i requisits tecnològics.

Pel que respecta a la viabilitat tecnològica, aquest projecte és viable ja que les Xarxes Neuronals tenen la potència suficient per a fer-ho. En aquest sentit, les Xarxes Neuronals Convolucionals les quals estan molt avançades en l'aplicació en la visió per computador i molts de treballs anteriors com serien [2], [3] o [4] ho demostren. A Internet hi ha disponibles una gran quantitat d'implementacions d'aquest tipus de xarxes que són de lliure ús, per tant no s'han d'implementar de zero. Pel que fa al requisit de poder fer el seguiment dels animals i les persones, en el camp de la robòtica s'han fet ús i desenvolupat molts d'algorismes per a dur-ho a terme ja que sempre s'ha mostrat la necessitat de fer seguiment d'aquests. Alguns d'aquests algorismes, com seria el Filtre de Kalman, són també aplicables a les seqüencies d'imatges, ja que seria com treballar en un món 2D. Al igual que amb la Xarxa Neuronal, també hi ha moltes implementacions d'aquest algorisme i les seves variacions disponibles a internet les quals són de lliure ús, per tant tampoc s'ha de implementar en la seva totalitat i només hi ha que modificar-los per a que compleixin els requisits. Tant la xarxa neuronal com l'algorisme de seguiment han de ser compatibles entre ells per a poder funcionar en conjunt. Algunes d'aquestes implementacions esmentades anteriorment són en Python, que és el llenguatge de programació elegit per a desenvolupar el projecte, per tant és factible el que es puguin combinar de forma senzilla.

Per a crear aquest sistema no només és necessari tenir la implementació si no també les dades per a poder fer proves. Pel que fa a les dades, hi ha disponibles grans quantitats de datasets de lliure ús d'imatges etiquetades de diferents objectes en xarxa, moltes de les quals contenen persones. També hi ha disponible

el dataset de porcs de la Universitat d'Edimburg [5] que només requereix citar-los per a poder fer-ne ús, per tant amb aquest ja es té coberta la part d'imatges de porcs.

Pel que fa al hardware necessari, l'única part que requereix d'una potència computacional fora dels limits d'un ordinador portàtil normal, del qual es disposa, és l'entrenament de la Xarxa Neuronal. Per a fer això, VICOROB té disponible un servidor que compta amb 376 GB de memòria RAM, 12 TB de memòria al disc dur i 2 GPUs de 24 GB de memòria cada una, per tant, es té capacitat computacional de sobra per a poder dur a terme el projecte i tenir una velocitat de processament suficient per a treballar còmodament. En aquest servidor es permet crear entorns de Conda [6], el que permet instal·lar i tenir disponible el compilador de Python i totes les llibreries necessàries per a la implementació.

2.2 Viabilitat i requisits econòmics

Pel que fa a les limitacions econòmiques, aquest projecte ha set desenvolupat en la seva totalitat amb recursos de lliure ús o cedits de forma gratuïta, per tant la implementació i les dades no tenen cap cost. Pel que fa al hardware, s'ha fet ús d'un servidor de la universitat per a desenvolupar en la seva totalitat el projecte i s'ha utilitzat l'ordinador de la mateixa estudiant per accedir a aquest, per tant no ha suposat cap gast afegit i ha fet que aquest projecte no necessiti de cap despesa addicional al material ja disponible. Això ha fet que el projecte hagi set viable econòmicament.

2.3 Viabilitat temporal

Si es parla de limitacions temporals, a la introducció ja s'ha indicat que s'ha hagut de limitar l'abast del projecte per ajustar-lo al temps disponible, per tant s'ha centrat en la detecció dels porcs i les persones i en l'algorisme de seguiment d'aquests. S'ha organitzat des de un principi la planificació del desenvolupament d'aquest projecte per a tenir suficient temps per a treballar totes les parts amb cura.

CAPÍTOL 3

Metodologia

Aquest projecte va ser una proposta del tutor Rafael Garcia. Inicialment es va presentar la idea exposada al capítol 1 de crear un sistema per a analitzar com afecta la presència de persones al comportament dels porcs. Seguidament, es va analitzar el que suposava dissenyar i implementar aquest projecte en la seva totalitat i es va arribar a la conclusió de que en el temps que es tenia per a desenvolupar el Projecte Final de Grau no es podia fer en la seva totalitat, per tant es va arribar al consens de només implementar la part de detecció de porcs i persones i l'algorisme de seguiment d'aquests, deixant fora dels marges del projecte la part de quantificar aquests moviments i fer l'anàlisi del comportament.

Durant tot el desenvolupament del projecte hi ha hagut comunicació contínua entre l'estudiant i el tutor en forma de reunions presencials i en línia i amb missatges de correu electrònic on s'ha anat informant dels avenços del projecte, s'han resolt dubtes i s'han discutit diferents tòpics referents al treball a fer.

Per a implementar aquest projecte han fet falta conèixer un mínim de conceptes sobre Aprenentatge Automàtic i algorismes de seguiment, exposats al capítol 5, els quals l'estudiant no tenia ja que no entren en els coneixements donats al llarg de la carrera, per tant, al inici del projecte, ha fet falta un període d'aprenentatge on s'han estudiat els diferents conceptes.

Quan ja es tenien els conceptes clars, el següent pas va ser fer l'anàlisi de les dades que es tenien i/o es necessitaven per al correcte desenvolupament d'aquest. Inicialment es tenia un dataset de porcs aconseguit per l'Universitat d'Edinburgh [5] utilitzat a [7] per a fer l'anàlisi del pressupost diari del temps dels porcs. Aquesta publicació només fa l'anàlisi del comportament dels animals mirant quan estan menjant, beguent, moguent-se... En el cas d'aquest projecte, s'analitza com afecta la presencia de persones al comportant dels porcs, mesurat en moviments, per tant, és necessari també tenir un dataset amb persones per a poder entrenar la detecció de persones i porcs. Per a fer això, s'ha utilitzat part del dataset de Google Open Images v4 [8], només fent ús d'imatges amb persones etiquetades. Al principi només s'anava a entrenar la detecció de porcs

i persones per separat ja que no es disposava d'un dataset que contingui's ambdós objectes a les seves imatges, però entrat en el desenvolupament d'aquest es va aconseguir un dataset proporcionat per l'IRTA [9] el qual conté seqüències d'imatges de persones interactuant en el mateix espai que els porcs, per tant va permetre poder fer entrenaments i proves amb ambdós junts. El problema amb aquest dataset és que no són imatges etiquetades, per tant l'alumne va haver d'emprar temps en etiquetar i processar algunes d'aquestes imatges per a poder fer-ne ús. Aquest fet fa que la quantitat d'imatges disponibles amb porcs i persones junts és més reduïda que de porcs i persones per separat. Les dades es varen repartir en tres datasets separats: entrenament, validació i testing.

Per a fer la detecció de persones i porcs es va haver d'elegir amb quin mètode fer-ho. Al analitzar les diferents opcions que es tenien, es va arribar a la conclusió de fer ús d'una Xarxa Neuronal Convolucional o CNN (concepte exposat a la secció 5.1.1), específicament un conjunt de CNNs específiques per a la detecció d'objectes anomenades Xarxes Neuronals Convolucionals basades en Regions o R-CNNs. Dintre d'aquest conjunt es va plantejar l'ús de dos R-CNNs diferents: Faster R-CNN [2] o YOLOv4 [3]. Després de fer proves amb ambdós i comparar diferents factors com a resultats i velocitat d'entrenament i detecció, es va arribar al consens de fer ús de la YOLOv4, ja que té un temps molt reduït de detecció, cosa que si en algun moment es vol extrapolar aquest sistema a un ús real, seria molt necessari. Quan es va tenir la xarxa elegida, es va començar a fer entrenaments amb les dades a partir dels pesos ja entrenats de la Darknet [10] i, a partir dels resultats obtinguts després de cada entrenament, es va anar refinant el dataset.

Per a fer el seguiment dels animals i les persones, des d'un principi el tutor va expressar la idea de fer ús d'un Filtre de Kalman (concepte exposat a la secció 5.2), ja que és un algorisme de seguiment àmpliament usat en el món de la robòtica que és relativament simple d'aplicar en visió per computador. Ja que el moviment que es té a les seqüències d'imatges no és lineal, es va haver d'elegir entre dos implementacions d'aquest aplicats a moviments no-lineals: Extended Kalman Filter o Unscended Kalman Filter (concepte exposat a la secció 5.2.1). Es va elegir finalment fer ús del Unscended Kalman Filter o UKF ja que els resultats són lleugerament millors que els de l'Extended Kalman Filter.

Quan ja es van tenir la Xarxa Neuronal i l'algorisme de seguiment elegits, es varen combinar, essent la sortida de la xarxa neuronal l'entrada de l'algorisme de seguiment (referir-se a 7 per a veure l'arquitectura d'aquest sistema). A mesura que s'anava entrenant la YOLOv4, s'anaven fent proves amb el UKF per veure com funcionaven en conjunt i anar refinant les dades i els paràmetres en

conseqüència.

Pel que fa a la memòria, al llarg del projecte s'ha anat fent un seguiment del procés i, quan ja es tenia implementat tot el sistema i només quedava refinar dades i paràmetres, es va començar a redactar aquest document en paral·lel a aquest refinament per a aprofitar el temps que tarden els entrenaments i les proves.

CAPÍTOL 4

Planificació

Com s'ha explicat al capítol 3, la feina a fer s'ha dividit en quatre blocs principals ordenats cronològicament, com es pot veure a la figura 4.1:

1. **Desenvolupament dataset**
    - Anàlisi i processament del dataset de porcs de la Universitat d'Edinburgh per a extreure les etiquetes de les imatges i posar les dades en el format necessari per a processar.
    - Anàlisi i processament del dataset de Google Open Images v4 per a extreure algunes de les etiquetes de les imatges de persones i posar les dades en el format necessari per a processar.
    - Anàlisi i processament del dataset de l'IRTA etiquetant algunes de les imatges proporcionades per a poder processar-les i fer-ne ús.
    - Durant tot el desenvolupament del dataset s'han creat i refinat tres datasets, entrenament, validació i testing, per entrenar i fer proves. A mesura que s'ha desenvolupat el projecte, s'han anat afegint i modificant algunes de les dades per a aconseguir millors resultats.
2. **Detecció de persones i porcs**
    - Per a poder desenvolupar la detecció, s'han hagut de conèixer alguns conceptes previs sobre l'Aprenentatge Automàtic, específicament sobre el Deep Learning.
    - Investigació i elecció de la Xarxa Neuronal a utilitzar per fer la detecció, essent finalment l'elecció la YOLOv4.
    - Cerca, anàlisi, modificació i refinament de la implementació de la YOLOv4 per a que sigui compatible amb el sistema sencer.
    - Entrenament dels pesos de la YOLOv4 per a aconseguir millors resultats. A mesura que es va entrenant, es va refinant el dataset, com s'ha dit al apartat 1.
    - Duta a terme de diferents proves per a veure el funcionament de la xarxa entrenada i analitzar si els resultats són els esperats i, si no ho eren, definir quins paràmetres del sistema hi havia que modificar i tornar a entrenar.

3. **Algorisme de seguiment**

    - Per a poder desenvolupar la detecció, s'han hagut de conèixer alguns conceptes previs sobre els Filtres de Kalman.
    - Investigació i elecció de la variant del Filtre de Kalman a usar, essent finalment el Unscented Kalman Filter per la seva aplicació a models no-lineals.
    - Cerca, anàlisi, modificació i refinament de la implementació de l'UKF.
    - Duta a terme de diferents proves en combinació amb la Xarxa Neuronal entrenada per a veure el funcionament funcionament en conjunt i fer anàlisi dels resultats i, si calia, fer modificacions a la part que pertoqui.

4. **Redacció de la memòria**

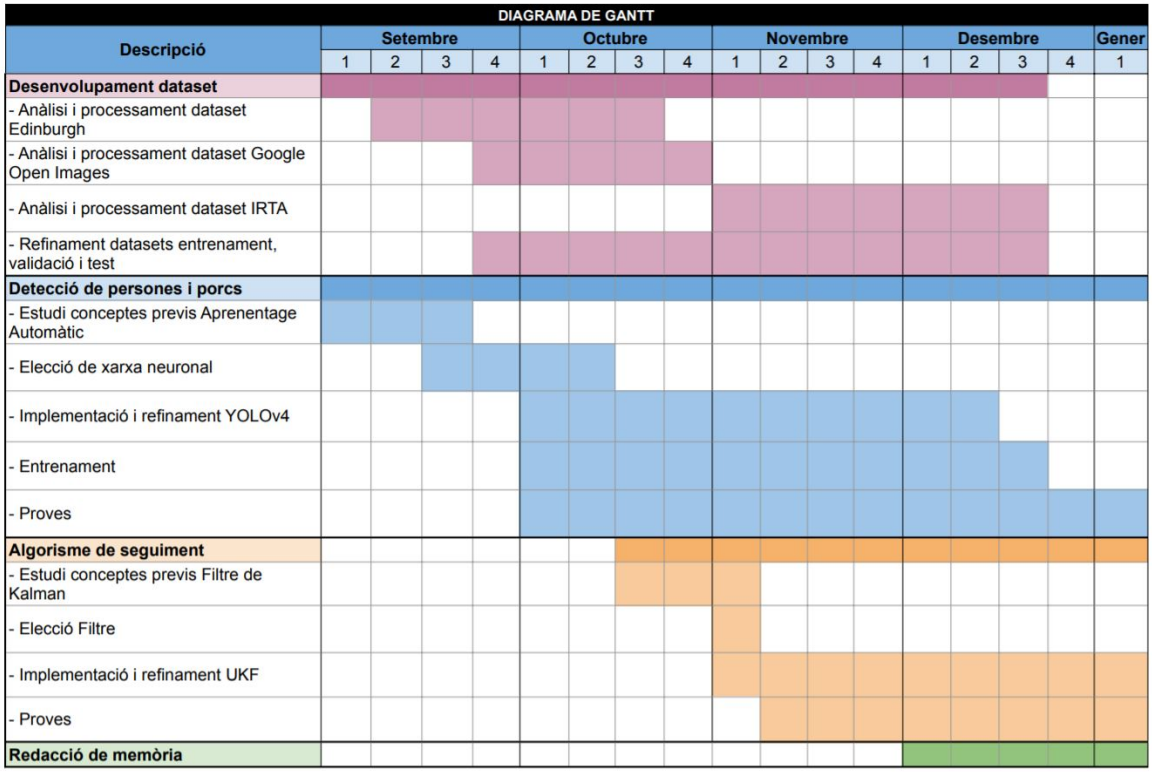


Figura 4.1: Diagrama de Gantt

CAPÍTOL 5

Marc de treball i conceptes previs

En aquest projecte només hi han col·laborat de forma directa la Virginia Ramón Ferrer (autora d'aquest document) amb l'assistència quan ha set necessària del tutor Rafael Garcia Campos.

Com ja s'ha esmentat prèviament, aquest projecte consisteix en fer ús d'una xarxa neuronal per a detectar la presencia de persones i porcs en vídeo i, seguidament, seguir la trajectòria d'aquests amb un algorisme de tracking. Per a entendre els continguts d'aquest, s'han de conèixer una sèrie de conceptes previs, que s'introdueixen a continuació.

5.1 Machine learning

Segons [11], l'aprenentatge automàtic (més conegut pel seu nom en anglès *Machine Learning*) es pot definir com a l'estudi sistemàtic d'algoritmes i sistemes que milloren el seu coneixement o rendiment amb l'experiència. Aquests algorismes construeixen un model basat en dades de mostra, coneguts com a dades d'entrenament, per tal de fer prediccions o decisions sense estar programats explícitament per fer-ho.

A vegades hi ha tasques per a les quals és complicat programar algorismes que indiquen a la màquina com executar tots els passos necessaris per resoldre el problema en qüestió, és a dir, sense que l'ordinador hagi d'aprendre. En aquests casos, sol ser més eficaç ajudar la màquina a desenvolupar el seu propi algorisme, fent així ús de tècniques de *Machine Learning* [12].

Segons la sortida que donen els algorismes de *Machine Learning*, es poden classificar en diferents tipus, dels quals els principals són [12]:

1. **Aprenentatge supervisat:** S'anomenen així als algorismes que generen una funció que relaciona cada entrada amb la sortida desitjada. Per a poder entrenar un algorisme supervisat, es necessiten dades que relacionen l'entrada amb la sortida, essent la sortida una classe, un valor... Aquest tipus de dades són conegudes com a dades "etiquetades".

2. **Aprenentatge no supervisat:** Al contrari que en el cas anterior, aquest tipus d'algorismes treballen amb dades "no etiquetades", és a dir, no es coneix la sortida desitjada, per tant, han de tenir la capacitat de trobar patrons en les dades per a ser capaç d'etiquetar-les.

3. **Aprenentatge semisupervisat:** Aquests algorismes són una combinació dels dos tipus anteriors, fent ús tant de dades "etiquetadesçom "no etiquetades"per tal de fer una classificació adequada.

4. **Aprenentatge per reforç:** Al contrari que ens casos anteriors on per cada entrada es volia una sortida, els algorismes d'aquest tipus són capaços d'avaluar i generar polítiques de seqüències d'accions, restant importància a les accions singulars.

5.1.1 Deep learning

L'aprenentatge profund (més conegut pel seu nom en anglès *Deep Learning*) forma part d'un conjunt de mètodes d'aprenentatge automàtic basats en xarxes neuronals artificials. L'aprenentatge pot ser supervisat, semisupervisat o no supervisat.

Una xarxa neuronal artificial (més coneguda com a ANN per les seves sigles en anglès *Artificial Neural Network*) és un algorisme de machine learning basat en el sistema nerviós dels animals, específicament dels èssers humans. Com es pot veure a la figura 5.1, una xarxa neuronal és un algorisme format per capes que a la seva vegada estan formades per unitats computacionals, conegudes com a neurones, les quals contenen una o més entrades i sortides ponderades i una funció de transferència que connecta d'alguna forma aquestes entrades i sortides. Una xarxa neuronal sempre conté una capa d'entrada, una capa de sortida i una o més capes "amagades" que duen a terme tots els càlculs necessaris per transformar aquesta entrada en la sortida desitjada [13].

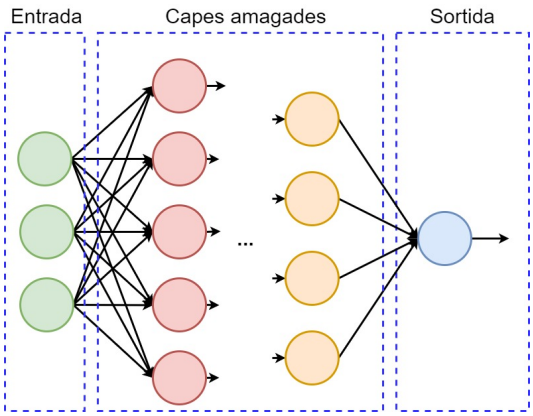


Figura 5.1: Arquitectura d'una xarxa neuronal artificial

Les xarxes neuronals poden ser classificades en diferents tipus segons el seu propòsit. Els tipus més comuns són:

1. **Xarxa neuronal directa:** Més coneguda pel seu acrònim anglès FNN (*Feedforward neuronal network*), les xarxes d'aquest tipus són les més simples, ja que les connexions entre nodes no formen cap cicle i la informació només es mou en direcció entrada a sortida.

2. **Xarxa neuronal convolucional:** Més coneguda pel seu acrònim anglès CNN (*Convolutional neural network*), és un tipus similar a la FNN, però fa ús de l'operació matemàtica coneguda com a "convolució", el que fa que aquesta classe de xarxa sigui molt utilitzada per al processament d'imatges. En aquest projecte es fa ús d'una CNN.

3. **Xarxes neuronals recurrents:** Més coneguda pel seu acrònim anglès RNN (*Recurrent neural network*), les connexions d'aquest tipus de xarxa poden formar cicles de realimentació entre neurones.

5.2 Filtre de Kalman

El Filtre de Kalman, normalment referit pel seu acrònim anglès KF (*Kalman Filter*), és un dels principals algorismes d'estimació. Aquest proporciona estimacions d'algunes variables desconegudes donades les mesures, o observacions, observades al llarg del temps. Els Filtres de Kalman han demostrat la seva utilitat en diverses aplicacions, també gràcies a que tenen una forma relativament simple i requereixen poca potència de càlcul [14].

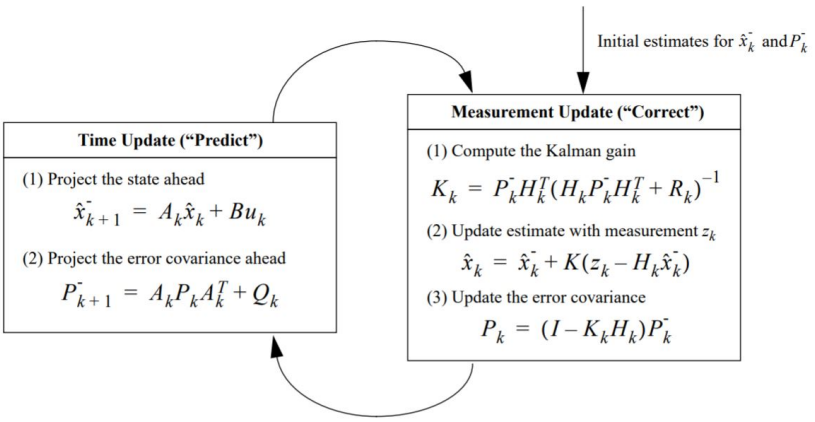


Figura 5.2: Operació del Filtre de Kalman [15]

Variable	Significat
$\widehat{x}_{\bar{k}}$	Projecció del estat
A_k	Matriu que descriu com l'estat evoluciona de k a k-1 sense controls o soroll
$\widehat{x}_k$	Actualització del estat
B	Matriu que descriu com el control u_k canvia l'estat de k a k-1
u_k	Vector d'entrada
$\widehat{P}_{\bar{k}}$	Projecció de la matriu de covariància
P_k	Matriu de covariància
Q_k	Matriu de covariància del soroll del sistema
K_k	Guany de Kalman
H_k	Matriu que descriu com mapejar l'estat $\widehat{x}_k$ a una observació z_k
R_k	Matriu de covariància del soroll de la mesura
z_k	Mesura

Taula 5.1: Taula explicativa de les variables de la figura 5.2

Aquest algorisme és un algorisme iteratiu que es du a terme en dos fases: la predicció i l'actualització, quan es tenen mesures. Aquestes fases es poden observar clarament a la figura 5.2 [15], on cada bloc és una de les fases. Si observem el primer bloc "Time Update ("Predict")", és a dir, la predicció, veiem que l'algorisme projecta l'estat futur $\widehat{x}_{k+1}$ i la matriu $P_{\bar{k}}$, que és la matriu de covariància del model. Seguidament, si es tenen mesures, a partir d'aquestes i de les projeccions fetes a la fase anterior es calcula el guany de Kalman K_k, s'actualitza l'estimació del estat $\widehat{x}_{\bar{k}}$ amb la mesura z_k i l'error de covariància P_k. Per a conèixer que és cada variable, referir-se a la taula 5.1.

Normalment es refereix com a Filtre de Kalman al que és el Filtre de Kalman Lineal, és a dir, l'estimador òptim de l'estat ocult quan es té un sistema dinàmic

lineal. Per a sistemes no lineals, com el que es presenta en aquest projecte, hi ha modificacions d'aquest mateix com serien l'Extended Kalman Filter (EKF) o l'usat en aquest projecte, donats els seus millors resultats en comparació amb el EKF, Unscented Kalman Filter (UKF). Aquestes modificacions aproximen els models no-lineals en models lineals.

5.2.1 UKF

L'UKF es basa en la intuïció de que és més fàcil aproximar una distribució gaussiana que no pas aproximar una funció o transformació no lineal arbitrària [16]. Aquest selecciona un nombre de mostres al voltant de la mitjana amb una tècnica de mostreig anomenada Unscented Transformation (UT) (Figura 5.3). Aleshores, els punts sigma es propaguen a través de la funció no lineal per a aconseguir un núvol de punts transformats, a partir de les quals es forma una nova estimació de mitjana i covariància.

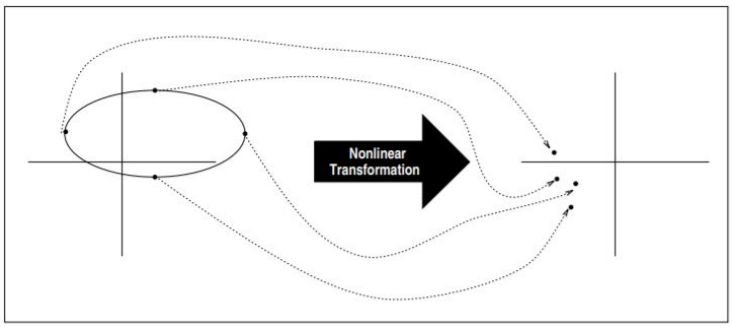


Figura 5.3: Exemple de l'UT [16]

CAPÍTOL 6

Estudi i decisions

En aquest capítol s'explica de forma detallada les decisions exposades al capítol 3 relacionades amb la implementació del sistema. En general, tot el sistema ha estat programat amb el llenguatge de programació Python [17], específicament en la seva versió 3.7 i programat, compilat i executat al servidor presentat al capítol 2.

6.1 Processament de dades

Pel que fa als datasets, tenim que, per a preprocessar-los per a poder fer-ne ús amb el sistema, s'ha fet ús de diferents scripts de Python. El dataset d'imatges de porcs de la Universitat d'Edimburg consta de videos de porcs amb etiquetes que estan guardades en fitxers en format JSON. Aquests han hagut de ser transformats al format que necessita el sistema, per tant, les seqüencies de vídeo han hagut de ser dividides en frames per a poder entrenar la xarxa neuronal, ja que només processa imatges individuals, i les etiquetes que estaven en format JSON han hagut de ser convertides a format text (aquests formats estan explicats al capítol 7). Pel que fa al dataset de persones, s'ha fet ús d'un Python Notebook creat per l'ususari de Github "RockyXu66" [18] "Object_Detection_DataPreprocessing.ipynb", el qual permet descarregar, processar i guardar les etiquetes del dataset Open Images v4. S'ha modificat per a només descarregar algunes de les imatges de persones i les seves etiquetes. Si es parla del dataset de persones i porcs de l'IRTA, també s'ha hagut de dividir els frames de les seqüències de vídeo com al dataset dels porcs, però, en aquest cas, les etiquetes han hagut de ser registrades a mà. Per a fer-ho, s'ha creat un script de MATLAB [19] el qual permetia indicar sobre les imatges les regions d'interès de cada classe i registrar aquestes dades a un fitxer de text en el format requerit.

Pel que fa als scripts de Python, s'ha fet ús principalment de les següents llibreries externes:

1. **NumPy**
    - *Versió*: 1.18.5

- *Descripció*: NumPy [20] és una biblioteca que afegeix suport per a vectors i matrius grans i multidimensionals, juntament amb una gran col·lecció de funcions matemàtiques d'alt nivell per operar en aquestes.
- *Ús*: En el cas d'aquest projecte, numpy es fa servir per crear i modificar matrius que guarden les dades necessàries per al funcionament del sistema, com serien les etiquetes de les imatges o paràmetres.

2. **CV2**

   - *Versió*: 4.1.1.26
   - *Descripció*: OpenCV [21] (Open Source Computer Vision Library) és una biblioteca de funcions de programació dirigides principalment a la visió per ordinador en temps real.
   - *Ús*: Aquesta biblioteca es fa servir per a processar les seqüencies d'imatges dels datasets i fer totes les funcions adients: llegir les imatges, retallar-les, afegir text, guardar imatges a la memòria...

3. **matplotlib**

   - *Versió*: 3.4.3
   - *Descripció*: Matplotlib [22] és una biblioteca completa per crear visualitzacions estàtiques, animades i interactives
   - *Ús*: S'usa per a mostrar per pantalla imatges i les seves etiquetes al processar les imatges de persones al Python Notebook.

4. **pandas**

   - *Versió*: 1.2.5
   - *Descripció*: Pandas [23] és una eina d'anàlisi i manipulació de dades de codi obert.
   - *Ús*: S'usa per a llegir els fitxers CSV que contenen les dades i les etiquetes de les imatges de les persones.

5. **scikit-image**

   - *Versió*: 0.17.2
   - *Descripció*: scikit-image [24] és llibreria que conté una una col·lecció d'algorismes per al processament d'imatges.
   - *Ús*: S'usa per a llegir les imatges de les persones que són a internet per la seva URL.

A part, també es fa ús principalment de les següents llibreries i/o mòduls del mateix Python:

1. **os**

    - *Descripció*: os [25] és un mòdul que permet fer ús de manera versàtil funcionalitats depenents del sistema operatiu, com seria obrir o llegir fitxers, carpetes...
    - *Ús*: Aquesta llibreria s'usa per a administrar fitxers, obrir, tancar, crear, llegir..., en aquest cas de text per a registrar i llegir les dades de les etiquetes.

2. **json**

    - *Descripció*: JSON és una llibreria que permet codificar i descodificar fitxers en format JSON (JavaScript Object Notation) [26].
    - *Ús*: Aquesta llibreria s'usa per a llegir els fitxers d'etiquetes de les dades del dataset de porcs de la Universitat d'Edimburg.

3. **random**

    - *Descripció*: Random [27] implementa generadors de nombres pseudoaleatoris per a diverses distribucions.
    - *Ús*: S'usa per a barrejar les imatges aleatòriament.

4. **shutil**

    - *Descripció*: Shutil [28] ofereix una sèrie d'operacions d'alt nivell sobre fitxers i col·leccions de fitxers, en particular, es proporcionen funcions que admeten la còpia i l'eliminació de fitxers.
    - *Ús*: S'usa per a copiar fitxers d'un lloc a un altre, en aquest cas, copiar imatges d'una carpeta general a unes altres dividides per datasets.

6.2 Detecció de persones i porcs

Pel que fa la detecció de les persones i els porcs, es va elegir fer ús de la xarxa neuronal convolucional YOLOv4 [3]. Es va elegir aquesta xarxa per la seva velocitat de processament d'imatges, el que permetria fer-ne ús en temps real. Pel que fa a la implementació, la implementació original d'aquesta, anomenada "darknet" [10], té l'arquitectura programada amb C i els programes per fer-ne proves en Python. Aquesta implementació és complicada d'entendre i, quan es

va intentar analitzar per poder fer-ne ús, es va veure que no era còmoda de re-entrenar, per tant es va buscar una implementació d'aquesta que fos en Python en la seva totalitat. Després d'analitzar varies opcions i fer algunes proves, es va decidir usar una implementació amb llicència d'ús MIT, per tant es pot usar lliurement mentre es doni crèdit al autor original del codi, implementada per l'usuari de Github "pythonlessons" [29]. Aquesta implementació està programada en la seva totalitat amb Python 3.7 i, a part d'algunes de les llibreries externes i llibreries i mòduls interns esmentades a la secció 6.1, fa ús principalment de la següent llibreria:

1. **tensorflow-gpu**
    - *Versió*: 2.3.0rc0
    - *Descripció*: Tensorflow és una biblioteca de programari obert per al aprenentatge automàtic [30], específicament per a construir i entrenar xarxes neuronals.
    - *Ús*: Aquesta llibreria s'usa per a crear l'arquitectura, entrenar i fer ús de la Xarxa Neuronal YOLOv4.

6.3 Algorisme de seguiment

Per l'algorisme de seguiment, es va decidir usar l'Unscented Kalman Filter per la seva aplicació en models no-lineals i el seu millor funcionament que el seu antecesor, l'Extended Kalman Filter. Després de analitzar si era millor programar-lo des del principi o si buscar una implementació ja feta, es va decidir per la segona opció ja que es tenia un temps limitat per a dedicar-li. Després de buscar diferents implementacions, es va trobar la implementació del usuari de Github "sj23patel" [31], el qual també feia ús d'una xarxa neuronal YOLO per a fer la detecció de la posició d'objectes, però només s'ha agafat la part del UKF, afegint-hi posteriorment la xarxa YOLOv4 esmentada a la secció anterior. Aquesta implementació està programada en Python també en la seva versió 3.7 en la seva totalitat. A part d'algunes de les llibreries externes i llibreries i mòduls interns esmentades a les seccions anteriors, fa ús també de les llibreries externes:

1. **sciPy**
    - *Versió*: 1.4.1
    - *Descripció*: SciPy [32] proporciona algorismes per a l'optimització, integració, interpolació, problemes de valors propis, equacions algebraiques, equacions diferencials, estadístiques i moltes altres classes de problemes.

- *Ús*: Aquesta llibreria s'usa per algunes de les seves funcions matemàtiques per a desenvolupar l'algorisme UKF.

Pel que fa a les llibreries i mòduls del mateix Python, fa ús de:

1. **threading**

   - *Descripció*: threading [33] és un modul que construeix una interficie d'alt nivell a sobre del mòdul _thread, el qual proporciona primitives a baix nivell per treballar amb múltiples threads.
   - *Ús*: S'usa per paral·litzar alguns dels processos i bloquejar recursos que han de ser accedits un a la vegada.

2. **copy**

   - *Descripció*: copy [34] aquesta llibreria proporciona operacions genèriques de còpia superficial i profundes d'objectes de Python, ja que les declaracions d'assignació d'aquest no copien objectes, sinó que creen enllaços entre un objectiu i un objecte.
   - *Ús*: S'usa per fer còpies dels valors d'objectes, per no sobreescriure'ls al modificar aquestes còpies.

3. **argparse**

   - *Descripció*: argparse permet definir i administrar els arguments que un programa requereix.
   - *Ús*: S'usa per definir els arguments que pot tenir un programa a l'hora de cridar-lo per línia de comandes.

CAPÍTOL 7

Anàlisi i disseny del sistema

En aquest capítol s'explica el disseny de cada part del sistema, específicament del processament de dades i la detecció de persones i porcs, i el disseny general del sistema, on també s'explica el funcionament del algorisme de seguiment. Al llarg del capítol hi ha diagrames, els quals estan codificats per colors: en blau allò dut a terme amb un programa de Python, en verd el dut a terme amb un programa de MATLAB i en vermell el dut a terme a mà.

7.1 Processament de dades

Com ja s'ha explicat a la secció 6.1 i es pot veure a la figura 7.1, es tenen tres datasets els quals han de ser processats per a poder fer-ne ús:

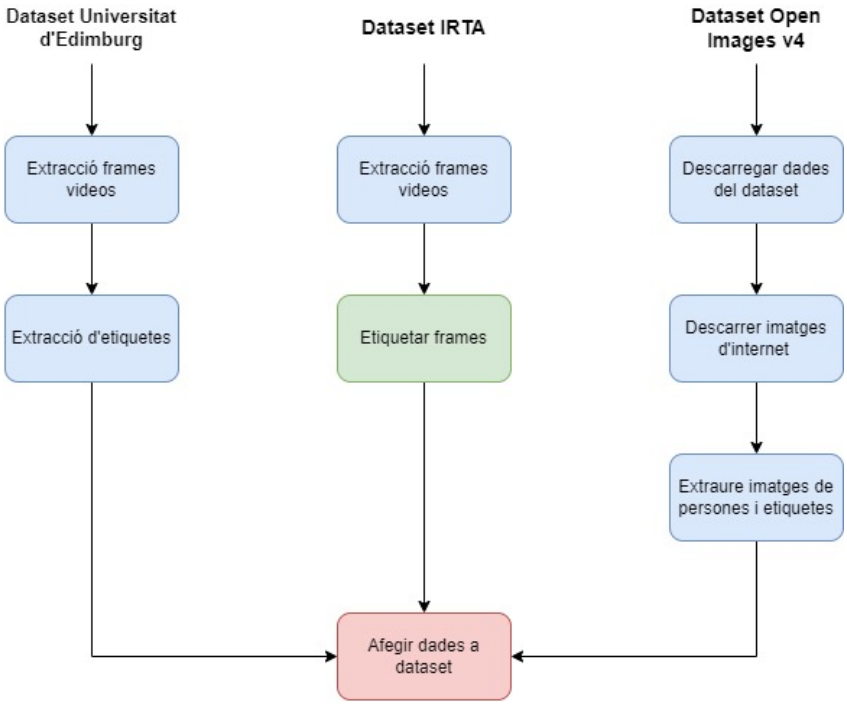


Figura 7.1: Diagrama del processament de dades

Aquests datasets han de ser convertits al format que es necessita per a entrenar i fer proves amb la xarxa neuronal. Aquest format, com es pot veure a la figura 7.2, consisteix en fitxers de text on cada línia correspon a una imatge etiquetada. Cada una d'aquestes conté el path absolut on la imatge està guardada a la màquina i, separat per espais, les coordenades i la classe de cada objecte etiquetat a la imatge. Aquestes etiquetes tenen el format [x_min,y_min,x_max,y_max,classe]. Com que només es treballa amb dos classes, s'ha arribat al consens de que la classe porcs serà l'indicador "0" i la classe persona serà l'indicador "1".

```
1 /home/virginia/TFG/PIGDATA2/frames_persona_cut/GH010013_zenital_Val_Pers_0.jpg
  ↪ 2,2,297,222,1
2 /home/virginia/TFG/PIGDATA2/frames_persona_cut/GH010013_zenital_Val_Pers_10.jpg
  ↪ 2,2,295,241,1
3 /home/virginia/TFG/PIGDATA2/frames/GH010013_zenital_Val_2010.jpg
  ↪ 609,757,1108,874,0 1089,820,1572,1015,0 1771,852,1921,948,0
  ↪ 1636,598,1921,822,0 1737,387,1921,621,0 1563,375,1774,489,0
  ↪ 1512,322,1774,394,0 1284,127,1377,261,0 1227,102,1285,163,0
  ↪ 1044,127,1261,244,0 993,204,1144,444,1
```

Figura 7.2: Exemple del format dels datasets

D'aquestes dades es generen tres datasets:

- *Entrenament:* Per entrenar la xarxa neuronal.
- *Validació:* Per a anar comprovant la correctesa els resultats generats per l'entrenament.
- *Testing:* Quan es té un resultat acceptable de validació, per comprovar que aquests resultats no són resultat del overfitting de la xarxa.

El dataset d'entrenament conté dades dels datasets de la Universitat d'Edimburg, de l'IRTA i del Open Images v4. Per la seva part, els datasets de validació i testing només contenen imatges de la Universitat d'Edimburg i de l'IRTA, ja que a la pràctica no ens interessa que la xarxa detecti coses com un vianant creuant el carrer, però sí que la xarxa tingui més dades per a conèixer el que és una persona, per tant, per validar i test amb les persones etiquetades de l'IRTA ja és suficient. A la taula de la figura 7.1 es pot observar quantes imatges de cada un dels datasets d'imatges s'utilitzen a cada dataset per a desenvolupar el model i a la figura 7.2 quantes etiquetes de persones i porcs hi ha de cada un dels datasets. Si s'analitzen aquestes dades, s'arriba a la conclusió de que s'ha

fet ús d'un 76% de les dades per a l'entrenament, un 13% per a la validació i un 11% per al testing.

		Dataset			Total
		Entrenament	Validació	Testing	
Imatges	Universitat d'Edimburg	4200	1395	1200	6795
	IRTA	2860	2595	1327	6792
	Open Images v4	8563	0	0	8563

Taula 7.1: Resum de les imatges usades dels datasets

			Dataset			Total etiquetes
			Entrenament	Validació	Testing	
Etiquetes	Universitat d'Edimburg	Porcs	33600	9600	9600	52800
	IRTA	Porcs	8234	2639	979	11852
		Persones	2971	1439	1388	5798
	Open Images v4	Persones	36563	0	0	36563
Total etiquetes			81368	13678	11967	

Taula 7.2: Resum de les etiquetes usades dels datasets

7.1.1 Dataset Universitat d'Edimburg (porcs)

Aquest dataset conté seqüències de vídeo amb imatges com es mostren a la figura 7.3, de les quals algunes estan etiquetades en fitxers JSON (un per cada vídeo). Aquest fitxers JSON contenen informació del vídeo com en nom del fitxer, el seu path o la seva configuració, com es pot veure a la figura 7.4, i un registre de la posició de cada porc al llarg dels frames, com es pot veure a la figura 7.5, on per cada frame on el porc és visible es guarda la posició i mida de la bounding box del porc, si és ground truth, si aquest és visible i que està fent (ja que, com em dit abans, aquest dataset era usat per a analitzar el pressupost de temps diari dels animals). En el cas de l'ús que se'n fa en aquest projecte, només es necessita la informació de la bounding box a cada frame dels porcs.

D'aquest dataset s'extreuen frames separats entre si de les seqüències de vídeo etiquetades (per exemple els frames múltiples de 10), els quals es guarden a un repositori de la mateixa màquina, i es converteixen les seves etiquetes al format esmentat prèviament en aquest capítol (figura 7.2).


Figura 7.3: Exemple de fotogrames del dataset de la Universitat d'Edimburg

```
"videoFileName":"color(4).mp4",
"fullVideoFilePath":
 ↪  "test_pigs_11_11_19/2019-11-11--11:34:50/000028/color.mp4",
"stepSize":0.1,
"config":{"stepSize":0.1,"playbackRate":0.4,"imageMimeType":"image/jpeg",
 ↪  "imageExtension":".jpg","framesZipFilename":"extracted-frames.zip",
 ↪  "consoleLog":false}
```

Figura 7.4: Informació del vídeo en fitxer JSON

```
{"frames":[{"frameNumber":0,"bbox":{"x":472,"y":81,"width":269,"height":145},
 ↪  "isGroundTruth":true,"visible":true,"behaviour":"sleep"},
 ↪  {"frameNumber":82,"bbox":{"x":469,"y":81,"width":273,"height":145},
 ↪  "isGroundTruth":true,"visible":true,"behaviour":"sleep"},,...}],"id":"1"}
```

Figura 7.5: Dades de les bounding boxes en fitxer JSON

7.1.2 Dataset IRTA (porcs i persones)

Aquest dataset conté seqüències de vídeo amb imatges com es poden observar a la figura 7.6, les quals no estan etiquetades.

Figura 7.6: Exemple de fotogrames del dataset de l'IRTA

Al igual que amb el dataset anterior, també s'extreuen i es guarden alguns dels frames d'aquestes seqüències, però, pel que fa a les etiquetes, han de ser creades. Aquestes etiquetes es creen amb un programa de MATLAB i es guarden directament al format de la figura 7.2.

Com que la proporció de persones i porcs a cada imatge és desproporcionada, ja que per cada persona hi ha una gran quantitat de porcs, s'han extret retalls d'alguns dels frames de persones per a poder tenir més quantitat d'imatges de persones. Al igual que el procés d'etiquetat, això també s'ha fet amb un script de MATLAB. A la figura 7.7 es pot veure algunes de les imatges resultants d'aquest procés.

Figura 7.7: Exemple de fotogrames de persones dataset de l'IRTA

7.1.3 Dataset Open Images v4 (persones)

Open Images v4 és un dataset d'aproximadament 9 milions d'imatges que s' han anotat amb etiquetes a nivell d'imatge, quadres delimitadors d'objectes i relacions visuals. El conjunt d'entrenament de v4 conté 14,6 milions de caixes delimitadores per a 600 classes d'objectes en 1,74 milions d'imatges, el que el converteix en el conjunt de dades més gran existent amb anotacions d'ubicació d'objectes [8]. A la figura 7.8 es poden observar algunes de les imatges de persones d'aquest. D'aquestes imatges, en aquest projecte només s'han extret 8562 imatges aleatòries amb etiquetes de persones per afegir al dataset d'entrenament. Per a fer-ho, hi ha hagut que descarregar les dades del dataset (URL de les imatges i les seves etiquetes), les quals estan registrades en fitxers .CSV, descarregar aquestes imatges a un repositori de la màquina i transformar les etiquetes d'aquestes al format de la figura 7.2.

Figura 7.8: Exemple de fotogrames de persones del dataset de Open Images v4

7.2 Detecció de persones i porcs

Per a desenvolupar el model de xarxa neuronal per a la detecció de persones i porcs, l'arquitectura de la qual està explicada a la secció 7.2.2, s'ha dut a terme el procés de la figura 7.9. Inicialment es varen definir uns datasets d'entrenament i validació amb imatges del dataset d'Open Images v4 i la Universitat d'Edimburg. Seguidament es varen definir els paràmetres a entrar a la xarxa i es va entrenar. Amb la validació es va fer un anàlisi dels resultats d'aquest entrenaments. El resultat no era dolent, però la detecció de persones no era la esperada. Paral·lelament va arribar el dataset de l'IRTA, el qual va permetre tenir ambdues classes en el mateix espai, per tant es va poder prescindir de les imatges d'Open Images v4 a la validació. Es va repetir aquest procés de millorar el dataset, afegint o canviant dades i paràmetres, entrenar i analitzar resultats fins a aconseguir un resultat acceptable, per a poder començar a treballar en conjunt amb l'UKF.

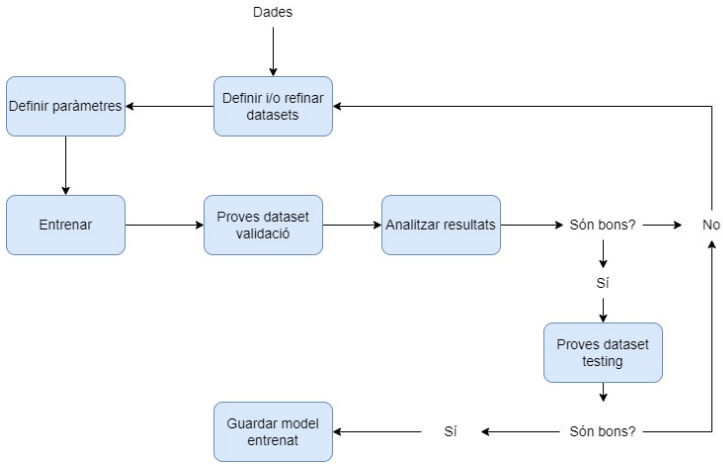


Figura 7.9: Diagrama del desenvolupament del model de la YOLOv4

A la figura 7.10 es poden observar com es mostren els resultats d'aquesta detecció: caixes blaves que indiquen la posició d'un objecte, amb el nom de la classe a la que pertany i la probabilitat amb la que ha identificat aquest objecte.

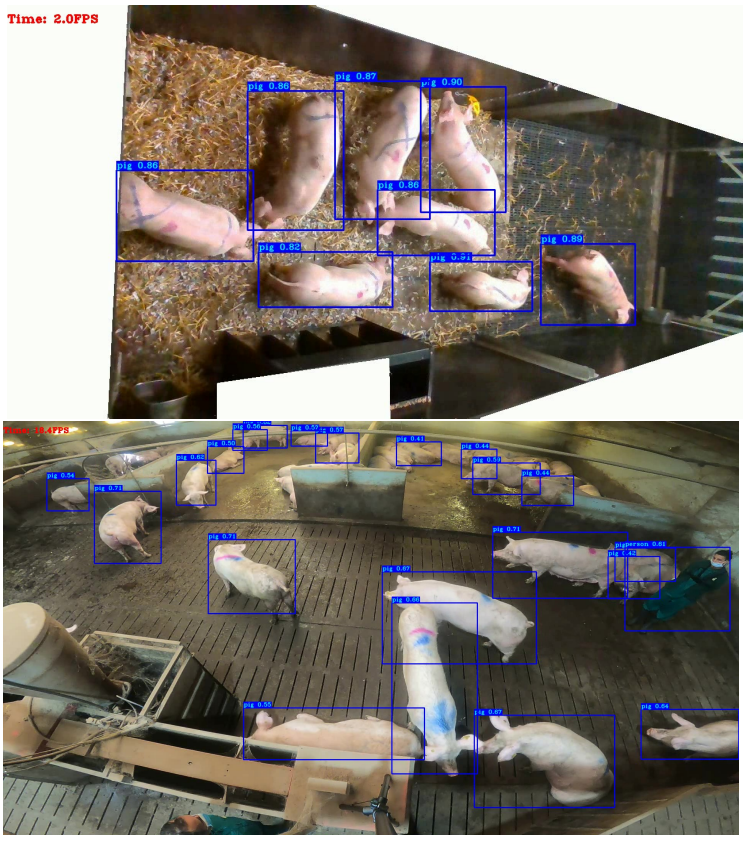


Figura 7.10: Exemple de resultats del model entrenat de la YOLOv4

7.2.1 Quantificació del error del model

Per quantificat l'error del model entrenat es fan ús de dos grups d'error o pèrdues:

1. *IoU:* Intersection over Union és una mètrica d'avaluació que s'utilitza per mesurar la precisió d'un detector d'objectes en un conjunt de dades concret [35]. Aquesta està basada en l'index de Jaccard que és una mesura de semblança i diversitat entre conjunts de mostres finits, com es pot veure figura 7.11. IoU fa ús d'aquest mètode però amb les àrees de dos regions d'interés, com es pot veure a la figura 7.12.

$$J(A,B) = \frac{A \cap B}{A \cup B}$$

Figura 7.11: Càlcul del index de Jaccard entre dos conjunts A i B

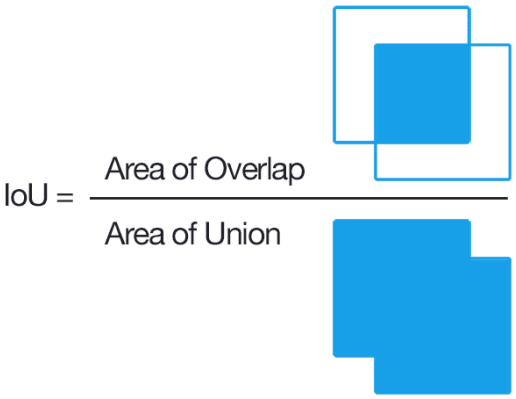


Figura 7.12: Càlcul del IoU [35]

2. *Pèrdues del entrenament:*

   (a) *GIoU loss:* GIoU és una millora d'IoU, la qual inclou la forma i l'orientació de l'objecte a més de l'àrea de cobertura. [36] van proposar trobar la regió d'interès d'àrea més petita que pugui cobrir simultàniament l'àrea d'interès de la predicció i l'àrea d'interès real, i utilitzar aquesta àrea com a denominador per substituir el denominador utilitzat originalment a l'IoU.

   (b) *Confidence loss:* Quantifica el segur que està la xarxa de la predicció que fa de la regió d'interès. S'espera que la predicció tingui una confiança d'1 quan hi ha objecte i de 0 quan no n'hi ha. Aquesta pèrdua quantifica quan dista la predicció d'aquests valors esperats.

(c) *Probability loss:* Quantifica el segur que està la xarxa de la predicció que fa de la classe de la regió d'interès. S'espera que la predicció tingui una probabilitat d'1 quan un objecte és d'una classe i de 0 de la resta de classes. Aquesta pèrdua quantifica quan dista la predicció d'aquests valors esperats.

(d) *Total loss:* Correspon a la suma dels tres errors esmentats anteriorment.

3. *mAP:* Average Precision (AP) o precisió mitjana és una mètrica d'avaluació per quantificar com de bona és la predicció de la xarxa en una classe comparant quantes prediccions correctes i incorrectes fa. Mean Average Precision (mAP) és el valor mitjà del AP d'un grup de classes.

7.2.2 Arquitectura de la YOLOv4

Com es pot observar a la figura 7.13, els models de detecció d'objectes es poden classificar en models d'una o dos etapes. Els models d'una etapa són capaços de detectar objectes sense necessitat d'un pas preliminar per a detectar regions d'interès a les imatges, a diferència dels models de dos etapes. Això fa que els models d'una etapa puguin treballar en temps real. En el cas d'aquest projecte es fa ús de la YOLOv4, model d'una etapa. Si s'observa una altra vegada la figura 7.13, es mostra que aquest tipus de model té una arquitectura en quatre parts:

1. *Input:* Poden ser imatges, piràmides d'imatges o pegats.

2. *Backbone:* És una xarxa neuronal profunda composta principalment per capes de convolució el objectiu principal de la qual és extreure les característiques essencials.

3. *Neck:* El rol principal d'aquest és recollir mapes de característiques de diferents etapes.

4. *Head:* És la predicció final, anomenada Dense Prediction, que es compon d'un vector que conté les coordenades del quadres delimitadors prevists o bounding boxes, la puntuació de confiança de la predicció i l'etiqueta.

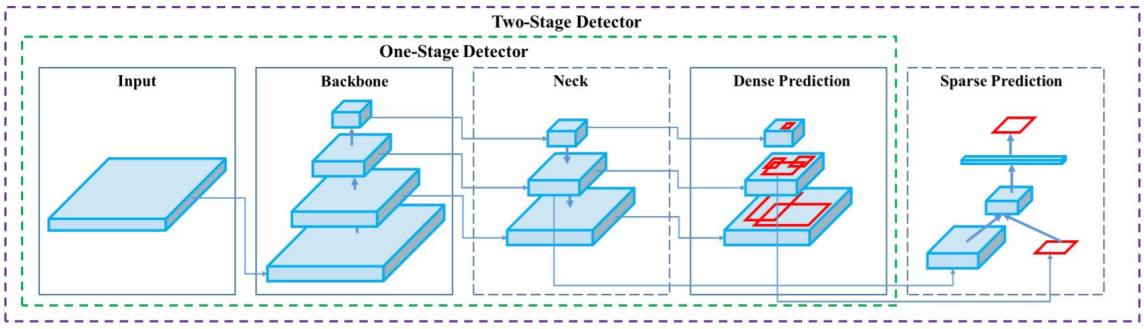


Figura 7.13: Object detector [3]

Si es parla específicament de la YOLOv4 utilitzada, aquestes quatre parts queden tal que:

1. *Input:* Imatges dels datasets de la secció 7.1

2. *Backbone:* S'utilitza la CSPDarknet53 [37] com a backbone.

3. *Neck:* En aquesta fase s'utilitza Path Aggregation Networks o PANet [38] concatenades amb mòduls de l'Spatial Pyramid Pooling o SPP [39].

4. *Head:* Per a la predicció es fa ús de la versió anterior d'aquesta xarxa, la YOLOv3.

7.3 Disseny general del sistema

Si es mira com funciona l'Unscented Kalman Filter en aquest projecte conseqüentment s'ha de parlar també de com s'ha acoblat la xarxa neuronal en aquest, per tant en aquesta secció es parlarà directament del funcionament general del sistema, és a dir, l'UKF en combinació amb la xarxa.

A la figura 7.14 es pot observar com funciona el sistema de dalt a avall:

1. Donada una seqüència de vídeo, per cada frame d'aquest es fa la detecció de persones i porcs amb el model entrenat de la YOLOv4.

2. Seguidament es comprova si el sistema tenia registrat algún track (s'entén per track el registre del moviment o trajectòria fet per un objecte) prèviament.

3. Si existeixen tracks:

(a) Per cada ROI (Regions d'interès o Regions of Interest en anglès, és a dir, les deteccions resultants) detectada es dibuixa la ROI al frame i es calcula l'IoU entre la ROI i l'última ROI registrada a cada track.

(b) Quan ja es tenen tots els IoUs de totes les ROIs i els tracks, s'assigna cada ROI nova amb un dels tracks sense repetir-les fent ús del algorisme hongarès. Aquest algorisme és un algorisme d'optimització per a la resolució de problemes d'assignació amb cost mínim [40].

(c) Si hi ha més ROIs noves detectades que tracks o hi ha deteccions no assignades, es crea un nou track per cada una d'aquestes deteccions no assignades.

(d) Si hi ha més tracks que ROIs detectades o hi ha tracks no assignats, per cada un d'aquests es comprova si du més d'un nombre definit de frames sense tenir nova assignació de ROI. Si és així aquest track s'elimina de la llista, altrament s'agafa l'última ROI registrada d'aquest track i es traslada seguint la predicció que fa el filtre de Kalman del moviment que ha pogut fer l'objecte del frame anterior a aquest.

4. Si no existeixen tracks, es crea un track per cada ROI detectada.

5. Finalment s'actualitza l'estat de cada track i es passa al següent frame.

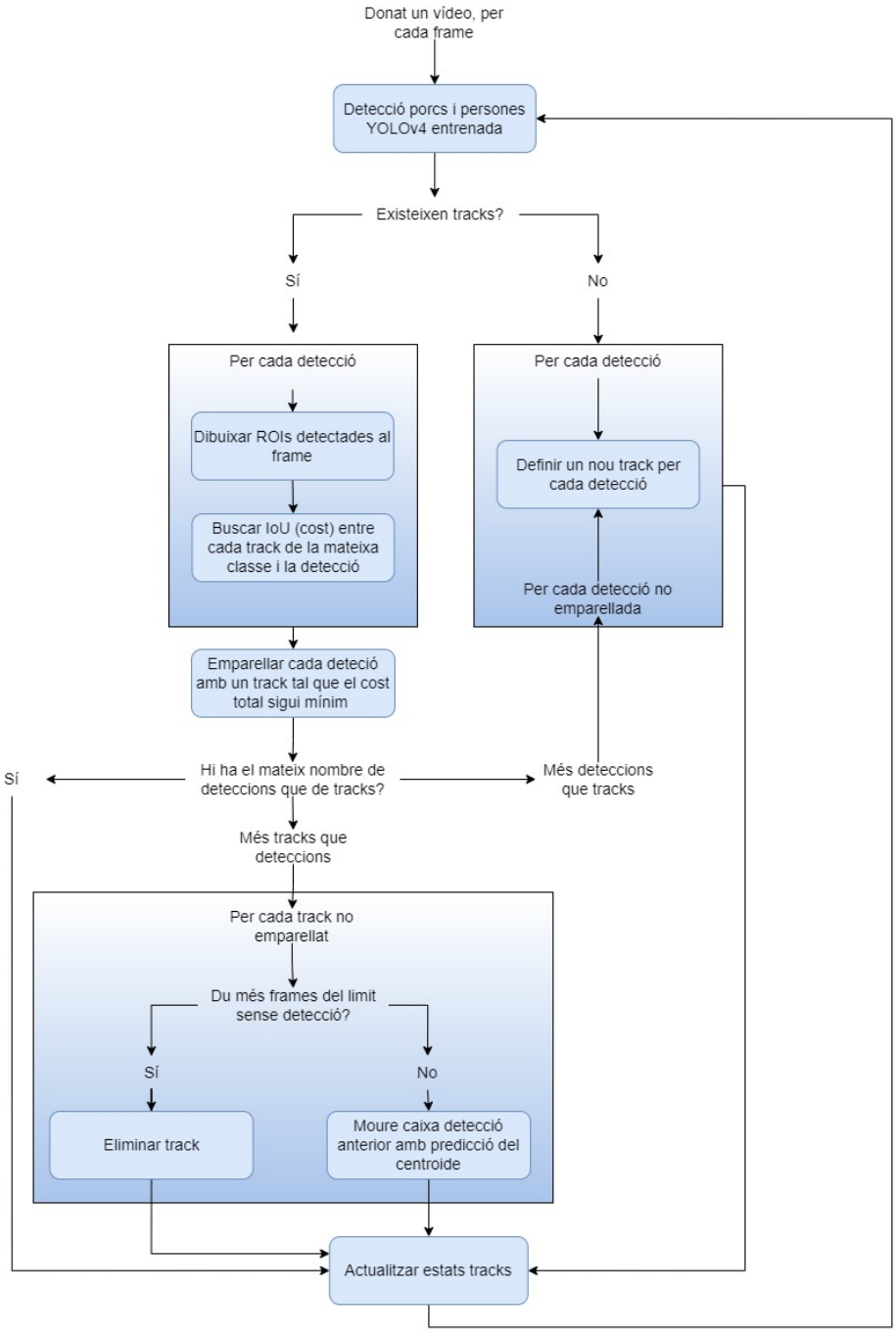


Figura 7.14: Diagrama del funcionament del sistema sencer

CAPÍTOL 8

Implementació i proves

En aquest apartat s'explicarà com s'ha implementat cada part del sistema i es presentarà com s'han dut a terme les proves de cada part. Els resultats finals d'aquestes proves es presentarà al capítol 9 de resultats.

8.1 Processament de dades

Si es torna a l'apartat 6.1, s'ha indicat com s'ha fet ús de cada dataset d'imatges de las que es disposava. En aquesta secció s'explicarà en detall com s'ha dut a terme el processament d'aquestes dades per a ser aptes per al seu ús en aquest projecte.

8.1.1 Implementació

8.1.1.1 Extracció de les dades del dataset de la Universitat d'Edimburg

Per extraure els frames i les etiquetes del dataset de porcs de la Universitat d'Edimburg s'ha creat un programa de Python que fa ambdues coses (annex B.2.1). Aquest programa processa tot el repositori de vídeos etiquetats d'una vegada. Tant per extraure els frames com les etiquetes, aquest programa recorr tots els subrepositoris d'un repositori d'anotacions i vídeos, per cada qual du a terme:

- *Extracció de frames:* Llegeix el vídeo i, per cada frame, aplica una màscara a la imatge (figura 8.1), ja que només es vol que la xarxa processi una part de la imatge, i guarda el frame a un repositori existent amb el nom del video i el número de frame que és.

- *Extracció d'etiquetes:* Per cada fitxer .JSON d'etiquetes (un per cada video) llegeix el fitxer sencer i guarda a una taula per cada frame les dades de cada porc que apareix en ella. Per fer això llegeix el registre de cada animal i, per cada frame al que surt, afegeix les dades a la taula. Aquestes dades es guarden en el format esmentat al capítol anterior generant un fitxer tipus text per cada vídeo.

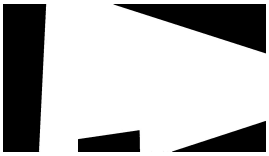
a)  b)  c) 

Figura 8.1: Processament frames Universitat d'Edimburg on: a) Frame, b) Màscara, c) Frame amb màscara aplicada

8.1.1.2 Extracció dels frames del dataset de l'IRTA

Com ja s'ha mencionat, s'han extret frames de les seqüències de vídeo dels datasets de l'IRTA per a poder entrenar la xarxa i fer proves. Per a fer això s'ha creat un programa de Python (annex B.2.2), el qual extrau frames d'una seqüència. Per a fer-ho, llegeix un vídeo ubicat a la màquina frame a frame, mantinguent un comptador general que du el seguiment del frame que s'està processant. Per cada frame, es comprova si aquest comptador indica que és un frame múltiple d'un valor (per no extreure frames molt seguits i evitar tenir la mateixa informació repetida), ja pot ser 10, 20, 30..., i, si ho és, guarda la imatge a un repositori existent amb el nom del video i el comptador en format .JPG.

8.1.1.3 Etiquetat de les dades del dataset de l'IRTA

Per a etiquetar els frames extrets de l'IRTA s'ha creat un codi de MATLAB (annex B.2.4) el qual du a terme els següents passos:

1. Llegir els fitxers que hi ha a un repositori entrat, per a tenir un registre de les imatges que hi ha.
2. Per a un grup d'aquestes imatges (ja que per a grans quantitats d'imatges és difícil etiquetar-les totes d'una):
   (a) Mostrar la imatge per pantalla.
   (b) Bucle que permet etiquetar tots els porcs a mà dibuixant un rectangle sobre la imatge per cada porc.
   (c) Bucle que permet etiquetar totes les persones a mà dibuixant un rectangle sobre la imatge per cada persona.
   (d) Guardar la informació de la imatge i les caixes creades per cada classe a un registre.
3. Guardar aquest registre de dades a un fitxer amb el format esmentat anteriorment.

8.1.1.4 Extracció de persones del dataset de l'IRTA

Per a extreure imatges de només persones, s'ha creat un codi de MATLAB (annex B.2.5) per a dur-ho a terme. Aquest codi té una estructura similar al anterior, ja que és una modificació d'aquest:

1. Llegir els fitxers que hi ha a un repositori entrat, per a tenir un registre de les imatges que hi ha.
2. Per a un grup d'aquestes imatges (ja que per a grans quantitats d'imatges és difícil etiquetar-les totes d'una):
   (a) Mostrar la imatge per pantalla.
   (b) Dibuixar una caixa al voltant de la persona a retallar.
   (c) Retallar i guardar imatge a un repositori.
   (d) Generar coordenades d'una regió d'interés, essent les messures de la imatge menys dos pixels (ja que la xarxa treballa amb regions d'interés).
   (e) Guardar la informació de la imatge i la caixa creada.
3. Guardar aquest registre de dades a un fitxer amb el format esmentat anteriorment.

8.1.1.5 Processament de les dades d'Open Images v4

Pel que fa al processament de les dades del Open Images v4, s'ha creat un Python Notebook (annex B.2.6) el qual du a terme els següents passos:

1. Descarrega i llegeix els fitxers .CSV que contenen les dades d'aquest dataset.
2. D'aquestes dades, s'extrauen totes les que contenen objectes etiquetats com a persona.
3. D'aquestes etiquetes es seleccionen un nombre n d'etiquetes aleatòries.
4. Es descarreguen a la màquina les imatges que contenen aquestes etiquetes i es divideixen en dos grups: per a entrenar i per a validar.
5. Per les imatges d'entrenament i validació:
   (a) Extraure les dades de les imatges i les etiquetes.
   (b) Guardar les dades a un fitxer amb el format necessari.

8.2 Detecció de persones i porcs

8.2.1 Implementació

Pel que fa a la detecció de persones, s'ha fet ús d'una implementació ja feta [29] en Python de la xarxa neuronal YOLOv4. Amb aquesta implementació s'ha entrenat i dut a terme proves per a comprovar els resultats d'aquest entrenament. Alguns d'aquests fitxers han set modificats per a complir els requisits del sistema.

8.2.1.1 Entrenament

L'entrenament de la xarxa s'ha fet amb un programa de Python "train.py" (annex C.3.5). Aquest programa fa ús d'un fitxer de Python que conté tots els paràmetres que s'usen a la configuració de la xarxa (annex C.3.1). Aquest programa consta de les següents etapes:

1. Carregar els pesos del model sobre el que es vol treballar, en aquest cas els pesos de la YOLOv4 Darknet [10].
2. Crear un fitxer de registre del procés d'entrenament, on per cada cicle d'entrenament es registra els valors del error.
3. Carregar els datasets d'entrenament i de validació.
4. Definir el nombre d'epochs (cicles d'entrenament) i la seva llargària.
5. Comprova si es vol entrenar des dels pesos de la Darknet, des d'un entrenament anterior o si es vol entrenar de zero i es crea la xarxa neuronal amb els pesos pertinents.
6. S'escull l'optimitzador a usar per a entrenar, en aquest cas l'optimitzador d'Adam, el qual és un mètode de descens de gradient estocàstic que es basa en l'estimació adaptativa de moments de primer i segon ordre. Segons [41] el mètode és "computacionalment eficient, té pocs requisits de memòria, invariant a la reescalada diagonal dels gradients i és molt adequat per a problemes que són grans en termes de dades/paràmetres".
7. Es defineix una funció "train_step" la qual du a terme un cicle d'entrenament del model amb un dataset d'entrenament. Això ho du a terme fent prediccions en grups d'imatges i es guarda l'error o loss a un registre i es fa el gradient d'aquest per a, amb l'optimitzador, aplicar un canvi petit als pesos del model per a intentar reduir l'error. Seguidament es calculen els errors presentats a "Perdues del entrenament" a la secció 7.2.1,es mostren, es guarden a un registre i es retornen.

8. Es defineix una funció ”validate_step” la qual du a terme un cicle de validació del model comprovant els resultats de fer prediccions amb el model entrenat a un dataset de validació. Això ho fa seguint el mateix mètode de la funció ”train_step” però sense aplicar cap canvi al model. Finalment també es calculen els errors, es mostren, es guarden a un registre i es retornen.

9. Per cada cicle d’entrenament:

   (a) Es fa un cicle d’entrenament amb la funció ”train_step” i es mostren els errors resultants.

   (b) Si hi ha dataset de validació es fa una validació dels resultats amb la funció ”validate_step”. Seguidament, donat els errors, es comprova si per paràmetres s’ha indicat si es volen guardar tots els models entrenats a cada cicle o el model amb menys error de validació. En el cas d’aquest projecte només es guarda el model amb menys error ja que guardar-los tots ocupa molta memòria.

10. Finalment, qual el model està entrenat, es comprova el valor del mAP per cada classe i en general per al model entrenat.

8.2.1.2 Predicció en imatges i seqüències de vídeo

Per a provar la predicció del model entrenat en imatges i seqüències de vídeo s’ha fet ús d’un programa de Python (annex C.3.6) el qual, donat una imatge o un vídeo emmagatzemat a la màquina, crea el model de la xarxa neuronal amb els pesos del entrenament i crida la funció ”detect_image” si rep una imatge o ”detect_video” si rep una seqüència de vídeo. Aquestes funcions generen una còpia del fitxer original de imatge que ara conté també les deteccions marcades i retorna el llistat de deteccions fetes i les classes a les que pertanyen. A la figura 8.2 es poden veure dos exemples de resultats.

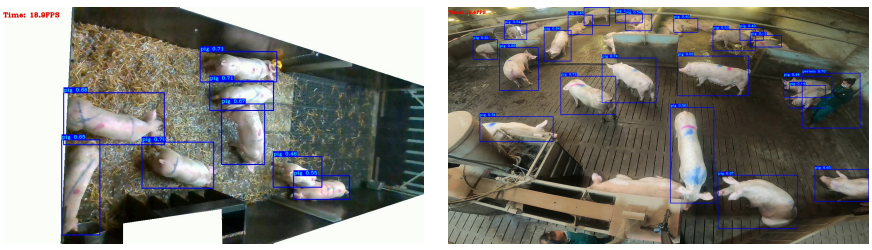


Figura 8.2: Processament frames Universitat d’Edimburg on: a) Frame, b) Màscara, c) Frame amb màscara aplicada

8.2.2 Proves

Per fer les proves s'ha fet ús d'un programa de Python que comprova el mAP de les prediccions d'un dataset entrat (annex C.3.7) i el programa per a la detecció presentat anteriorment. En les següents proves es farà ús dels següents termes:

- *Score threshold:* Aquest valor correspon a la probabilitat mínima de confiança de la predicció per a ser acceptada.

- *IoU threshold:* Aquest valor correspon al valor mínim que han de tenir dos regions d'interès d'IoU per a identificar-les com a una mateixa detecció, és a dir, un mateix objecte.

Aquestes proves han servit per a refinar els datasets i els paràmetres utilitzats a l'entrenament. A l'hora de modificar paràmetres d'una xarxa neuronal, prioritariament es modifiquen els següents:

1. *Learning Rate (LR):* La tasa d'aprenentatge o Learning Rate defineix la mida del pas a cada iteració que fa l'entrenament mentre es mou cap a un mínim d'una funció de pèrdua.

2. *Mida del batch:* La mida del lot o batch és el nombre de mostres processades abans d'actualitzar el model durant l'entrenament.

3. *Nombre de capes i neurones de la xarxa neuronal.*

Com que en aquest projecte s'ha fet ús de la YOLOv4, la qual ja té aplicació en la detecció d'objectes i està demostrat que funciona bé per a aquest fi, s'ha decidit centrar-se en fer proves amb el LR i la mida del batch, deixant de costat fer modificacions de l'arquitectura.

Si es parla de la mida del batch, s'han fet diferents proves amb 1, 2, 4 i 6 mostres per cada iteració. Es va arribar a la conclusió de que 1 i 2 no era un bon nombre per al batch ja que el temps de processament de cada iteració resultarà en un temps molt alt per cada entrenament. Al provar 4 i 6 es va veure que ambdós podrien ser bones opcions com a batch size ja que reduïa el temps d'entrenament notablement i els resultats eren bons. Finalment es va decantar per 4, ja que 6 donava lleugerament pitjors resultats en algunes proves realitzades.

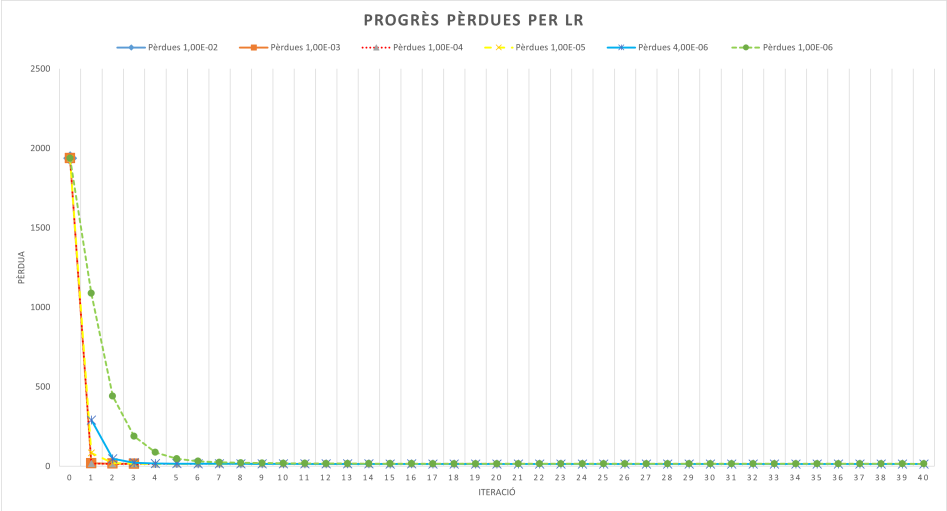


Figura 8.3: Gràfic de la comparació del progrés de la pèrdua total al llarg dels entrenaments amb diferents Learning Rates

Pel que fa al Learning Rate, s'han fet diferents entrenaments per a comparar com progressen els resultats comparant les pèrdues totals de cada iteració per a uns quants LRs amb valors entre 1e-6 a 1e-2. Si s'observa la figura 8.3 i la taula 8.1 es pot veure que com, amb valors petits de LR com 1e-6 tarda bastants epochs en arribar al mínim, en comparació a 4e-6, que tarda 6 epochs en comparació als 40 que tarda 1e-6 a arribar a valors al voltant del 15. Si es passa a 1e-5 s'arriba en 4 epochs i amb 1e-4 s'arriba en 2. Amb valors grans com 1e-3 o 1e-2 s'observa que en els primers epochs té error "nan" el que vol dir que hi ha hagut inconsistències numèriques, com seria dividir per 0 qualsevol valor, per tant es demostra que valors grans no són una bona elecció. Entre 1e-4 i 1e-6 es veu que més de la meitat de l'entrenament es queda al voltant dels 14 o els 15 i avança molt lentament.

Aquestes proves han servit per a definit els paràmetres a utilitzar a l'hora de fer les proves per als resultats finals.

		Learning Rate					
		1e-2	1e-3	1e-4	1e-5	4e-6	1e-6
Epoch	0	1938.35	1938.35	1938.35	1938.35	1938.35	1938.35
	1	nan	19.004	17.451	84.404	290.464	1109.10
	2	nan	16.881	15.465	21.109	47.421	454.646
	3	nan	16.065	15.272	16.492	22.974	187.491
	4	nan	nan	14.715	15.312	17.689	86.318
	5	nan	nan	15.099	14.913	16.359	47.679
	6	nan	nan	15.218	15.153	15.969	32.311
	7	nan	nan	15.532	14.681	15.678	25.756
	8	nan	nan	15.215	14.851	15.812	22.828
	9	nan	nan	15.729	14.789	15.400	21.129
	10	nan	nan	15.319	14.764	15.524	20.049
	11	nan	nan	16.078	14.790	15.269	19.348
	12	nan	nan	15.389	14.594	15.121	18.825
	13	nan	nan	15.655	14.742	15.084	18.508
	14	nan	nan	15.429	14.691	14.897	18.292
	15	nan	nan	15.706	14.746	14.833	17.997
	16	nan	nan	15.125	14.694	14.909	17.898
	17	nan	nan	15.268	14.755	14.738	17.832
	18	nan	nan	15.607	14.737	14.660	17.713
	19	nan	nan	15.951	14.689	14.771	17.674
	20	nan	nan	15.894	14.712	14.633	17.583
	21					14.621	15.975
	22					14.579	15.892
	23					14.688	15.790
	24					14.637	15.799
	25					14.654	15.740
	26					14.602	15.687
	27					14.566	15.636
	28					14.554	15.645
	29					14.537	15.589
	30					14.556	15.624
	31					14.597	15.619
	32					14.557	15.608
	33					14.542	15.571
	34					14.580	15.614
	35					14.567	15.606
	36					14.617	15.558
	37					14.575	15.598
	38					14.600	15.574
	39					14.545	15.539
	40					14.521	15.514

Taula 8.1: Progrés del loss de cada epoch dels diferents entrenaments donat el seu Learning Rate

8.3 Disseny general del sistema

8.3.1 Implementació

Pel que fa a la implementació general del sistema, formada per la detecció amb la YOLOv4 i l'algorisme UKF, s'explicarà com s'ha implementat el procés mostrat a la figura 7.14. Aquesta implementació és una modificació d'una implementació ja feta [31] programada en Python. En aquesta implementació s'ha modificat la detecció per fer-la amb la implementació explicada anteriorment, l'associació de ROIs i tracks s'ha canviat d'associació per distància dels centroides d'aquests (amb la Mean Squared Distance) a millor IoU, s'ha afegit la comprovació de que la ROI i el track tinguin la mateixa classe i que els tracks tinguin un registre de ROIs del objecte. Aquesta segueix els següents passos:

1. Inicialment, al fitxer "object_tracking.py" (Annex D.4.3), es defineix el path de la seqüència de vídeo la qual analitzar, és a dir, fer la detecció i el seguiment, i el path on guardar la seqüència resultant.

2. Es crea el model de la YOLOv4 per a fer la detecció.

3. S'inicialitza el registre de tracks amb la classe "Tracker" definida al fitxer "tracker.py" (Annex D.4.2). Aquesta classe s'encarrega de definir cada registre i aplicar-ne l'UKF.

4. Per cada frame de la seqüència de vídeo:

   (a) Es fa la detecció de porcs i persones fent ús de la funció "detect" definida al fitxer "frame_detection.py" (Annex D.4.5):

   - Aquesta funció rep una imatge i una xarxa neuronal YOLOv4 i crida la funció "detect_frame" definida al fitxer "utils.py" (Annex C.3.3) introduït anteriorment la qual fa la detecció de les ROIs de la imatge amb el model rebut i retorna un llistat de ROIs i la llista de classes.
   - Seguidament mostra el temps que ha tardat en fer la detecció i el nombre de deteccions resultant.
   - A continuació dibuixa sobre la imatge totes les ROIs i indica de quina classe és.
   - Finalment retorna la llista de centroides de les ROIs detectades i les mateixes ROIs.

   (b) Seguidament es fa l'actualització dels tracks amb les noves ROIs. Aquesta actualització es fa amb una funció de la classe "Tracker" introduïda, "Update":

- Si no existeixen tracks, es crea un track nou per cada ROI detectada.
- Si existeixen tracks:
  - Per cada ROI i cada track es calcula l'IoU entre ells, específicament 1-IoU ja que l'algorisme d'associació, l'algorisme Hongarès, busca el menor valor de cost i l'IoU dona valor més gran quant més coincideixen.
  - Quan ja s'han associats les ROIs i els tracks es comprova si hi ha tracks sense assignació o si alguna de les assignacions té un valor d'IoU fora del rang acceptat i passen a ser no assignats.
  - En aquests tracks no assignats es comprova si duen més d'un nombre definit de frames sense detecció i, si és el cas, s'elimina el track del registre.
  - Es comprova si hi ha ROIs sense assignar i, si és el cas, es crea un nou track per cada un d'aquests.
  - S'actualitza el Filtre de Kalman de cada track, en aquest cas del UKF. En aquest projecte l'UKF està definit com a una classe "UKF" definida en el fitxer "ukf.py" (Annex D.4.1), la qual té definides les funcions "predict" i "update" que fan la predicció i l'actualització del filtre seguint les equacions explicades a la secció 5.2.1.
  - Per cada track es crida la funció "predict" la qual du a terme un "prediction step", que bàsicament consisteix en predir el moviment que l'objecte ha dut a terme en una quantitat de temps indicada des de l'última predicció. Inicialment propaga els punts sigma (les mostres) de la funció no lineal de moviment amb una funció "iterate_x". A partir d'aquestes mostres es calcula l'estat "x_out", que passa a ser l'estat "x", i la nova covariància "p_out" a la qual se li afegeix el soroll del procés "q". Aquesta "p_out" calculada passa a ser la nova "p".
  - Quan ja ha fet la predicció, crida a la funció "update" per dur a terme l'actualització del estat i aconseguir l'estimació real del estat "x" donada la mesura del centroide de la ROI assignada, si en té.
- Quan ja es té el nou estat "x" i conseqüentment la predicció del desplaçament fet des del frame anterior al actual, si el track actual no té mesura, és a dir, nova ROI, es desplaçarà l'última ROI

registrada d'aquest seguint la posició predita pel filtre per a poder calcular l'IoU "correctament" al següent frame.

(c) A continuació, per cada track, es dibuixa sobre el frame la línia del moviment fet l'últim nombre definit de frames de la seqüència (només per veure el moviment de cada objecte sense plenar tota la imatge de línies les quals no es distingeixen).

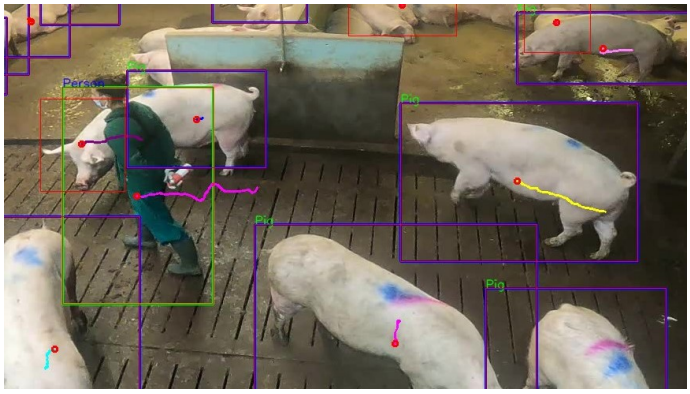
(d) Finalment es guarda el frame amb totes les annotacions fetes a la seqüència de vídeo resultant.

8.3.2 Proves

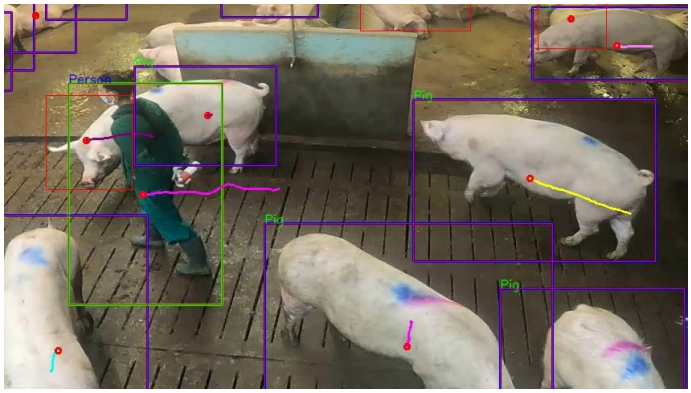
Pel que fa a les proves, s'ha fet ús del fitxer "object_tracking.py". Aquest filtre no ha necessitat de moltes proves ja que inicialment ja funcionava i la part que més afecta als resultats que dona és la detecció explicada anteriorment, per tant quasi totes les proves han set de la detecció en si.

Algunes de les proves fetes han set per a definir el valor de la matriu de covariància del soroll de la mesura "R" del fitxer "tracker.py" per a fer el filtre més "tranquil", ja que les prediccions d'aquests depenen molt de les deteccions de la xarxa i, com s'ha esmentat, aquestes encara són molt variables. Si s'observa la figura 8.4, es pot veure clarament l'efecte d'augmentar el valor d'aquesta matriu, ja que si es comparen les imatges a, b i c, en ordre de menor a major valor R, es pot veure com les trajectòries dels porcs i la persona observada són més lineals a mesura que es va augmentant aquest valor. Quant més gran és aquest valor, però, la predicció es retarda més, és a dir, prediu que està una mica més enrere del que realment està, per tant no es pot fer aquest valor tan gran com es vol.

a)



b)



c)

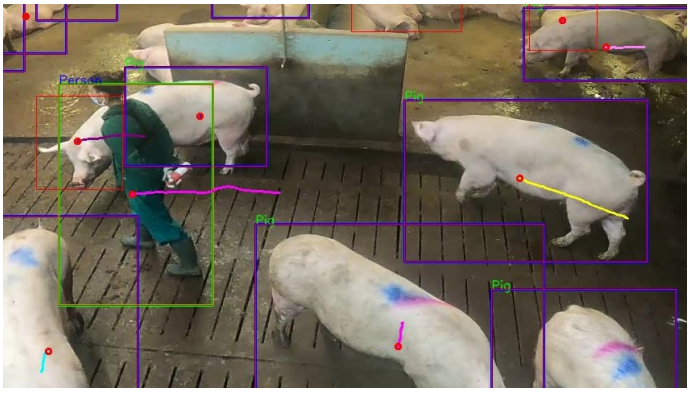


Figura 8.4: Comparació de la trajectòria predita pel UKF per a tres valors de la matriu R: a) 0.1, b) 1.0 i c) 2.5.

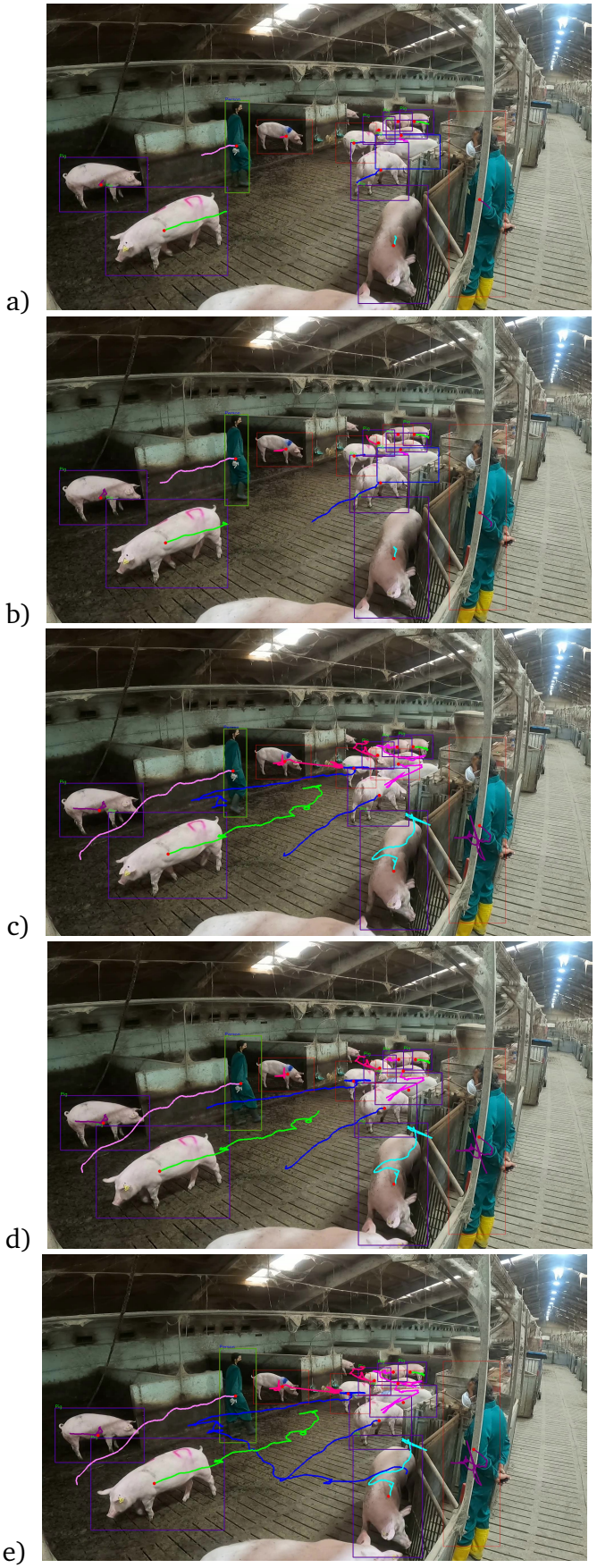


Figura 8.5: Visualització de llargària de trajectòries: a) 50, b) 100, c) 1000, d) 5000, e) Tota

A la vegada, s'ha definit un valor que restringeix la llargària de la trajectòria mostrada de cada objecte, en frames, ja que mostrar tota la trajectòria de cada objecte a la llarga fa que siguin indistingibles. S'han fet algunes proves comparant com es visualitzen les trajectòries per buscar un valor suficientment llarg per a poder seguir la trajectòria còmodament. A la figura 8.5 es pot observar la comparació de mostrar diferents llargàries de trajectòries. L'imatge a) mostra 50 frames de trajectòria, la b) 100, la c) 1000, la d) 5000 i la e) tota la trajectòria. En aquesta seqüència d'imatges en concret es pot veure com la c), la d) i la e) no són viables ja que no deixen visualitzar correctament totes les trajectòries, per tant, si es volen veure totes les trajectòries a la vegada tenint un nombre alt d'objectes i poder treure alguna conclusió, el millor valor és un valor entre 50 i 100.

A la secció 9.2 es presentaran els resultats finals de les proves i els errors més comuns donats. Aquestes proves s'han realitzat amb un valor d'R de 2.5 ja que dona trajectòries més llises i no retarda massa la trajectòria.

CAPÍTOL 9

Resultats

9.1 Detecció de persones i porcs

Com ja s'ha introduït en aquest document, es tenen dos datasets que contenen porcs, el de la Universitat d'Edimburg i el de l'IRTA, el primer dels quals es té una quantitat decent de dades etiquetades i amb un punt de vista de la càmera bastant beneficiós a l'hora de diferenciar els animals, mentre que el segon es caracteritza per ser dades menys controlades i marcades per la falta d'etiquetes. Aquesta diferència és visible al llarg dels resultats obtinguts.

La xarxa neuronal final ha set entrenada amb un LR de 5e-6, durant 50 epochs, amb un batch de 4. A la taula 9.1 es poden observar els resultats obtinguts a les diferents proves. Com es pot observar, es fa ús de tres score thresholds diferents i de quatre datasets:

- *Universitat d'Edimburg:* Dataset on només hi ha les imatges de validació i test del dataset de la Universitat d'Edimburg.
- *IRTA:* Dataset on només hi ha les imatges de validació i test del dataset de l'IRTA.
- *Dataset de validació*
- *Dataset de test*

Si es comparen els resultats entre els dos primers datasets en la classe porc, es veu que hi ha una diferència molt alta d'AP per a tots els score thresholds utilitzats, arribant a tenir una diferència de més del 30%. En ambdós cassos l'AP es redueix notablement quan més gran es fa l'score threshold, passant del 92.55% al 26.32% al dataset de la Universitat d'Edimburg i del 59.56% al 14.21% al dataset del IRTA només amb una diferència de 0.2 de valor Si es miren els resultats de les persones de l'IRTA, es veu que són molt millors que els dels porcs, variant menys d'un 10% entre l'score threshold del 0.5 al 0.7, tinguent 96.56% i 86.77% respectivament. Això es dona perquè les persones en aquest dataset duen sempre una indumentària similar, per tant ha pogut generalitzar-les millor que els porcs, els quals tenen marques pintades i tendeixen a estar molt junts. Si

es miren els datasets de validació i de test, es veu que els resultats són similars als del porc de la Universitat d'Edimburg, ja que una gran part de les mostres que es tenien venia d'aquest dataset, però l'AP es veu penalitzat per la presència d'imatges del IRTA pel que fa als porcs.

		Dataset								
		Universitat d'Edimburg	IRTA			Val			Test		
AP (en %)		Porc	Porc	Pers.	mAP	Porc	Pers.	mAP	Porc	Pers.	mAP
Score thres.	0.5	92.55	59.56	96.56	78.06	87.31	96.87	92.09	87.57	96.25	91.91
	0.6	68.76	37.74	93.07	65.40	61.76	93.33	77.54	66.37	92.80	79.58
	0.7	26.32	14.21	86.77	50.49	21.20	86.73	53.96	28.17	86.82	57.49

Taula 9.1: Resultats proves mAP als datasets en percentatges

Si es fa un anàlisi més exhaustiu dels resultats anteriors i es miren les deteccions que ha fet la xarxa a cada prova, com es pot veure a la figura 9.2, es reafirma el que ja s'havia dit anteriorment de que al model li costa més detectar els porcs del dataset del IRTA que del dataset de la Universitat d'Edimburg tot i que d'aquest últim també es veu que hi ha porcs que no ha detectat però proporcionalment molt menor. Si s'observen aquestes dades també es pot veure com en alguns casos també es detecten animals i persones que, en teoria, no hi son a les imatges, sobre tot en els porcs. En el cas de les imatges del IRTA, moltes d'aquestes deteccions en realitat no són deteccions errònies, si no que a l'hora d'etiquetar a mà aquestes dades, sobre tot en situacions on molts de porcs estan molt junts i és difícil diferenciar-los inclús amb la vista humana afectat també per la distancia dels porcs i la càmera i la qualitat de les imatges, hi ha hagut regions no etiquetades **on realment sí que hi ha porcs o alguna persona**, com es pot veure exemplificat a la figura 9.1, on es compara les deteccions fetes (en blau les deteccions que coincideixen amb el ground truth i en verd les que no apareixen al ground truth de la imatge), amb el ground truth (en vermell les ROIs que no s'han detectat).

U. Edim. Porcs 0.5		GT	
		P	N
Pred	P	17955	243
	N	1245	

U. Edim. Porcs 0.6		GT	
		P	N
Pred	P	13334	144
	N	5866	

U. Edim. Porcs 0.6		GT	
		P	N
Pred	P	5108	59
	N	14092	

IRTA Porcs 0.5		GT	
		P	N
Pred	P	2208	194
	N	1410	

IRTA Porcs 0.6		GT	
		P	N
Pred	P	1376	50
	N	2242	

IRTA Porcs 0.7		GT	
		P	N
Pred	P	514	0
	N	3104	

IRTA Pers. 0.5		GT	
		P	N
Pred	P	2730	8
	N	97	

IRTA Pers. 0.6		GT	
		P	N
Pred	P	2631	1
	N	196	

IRTA Pers. 0.7		GT	
		P	N
Pred	P	2453	0
	N	374	

Val. Porcs 0.5		GT	
		P	N
Pred	P	10759	199
	N	1480	

Val. Porcs 0.6		GT	
		P	N
Pred	P	7595	69
	N	4644	

Val. Porcs 0.7		GT	
		P	N
Pred	P	2599	13
	N	9640	

Val. Pers. 0.5		GT	
		P	N
Pred	P	1394	5
	N	45	

Val. Pers. 0.6		GT	
		P	N
Pred	P	1343	1
	N	96	

Val. Pers. 0.7		GT	
		P	N
Pred	P	1248	0
	N	191	

Test Porcs 0.5		GT	
		P	N
Pred	P	9404	238
	N	1175	

Test Porcs 0.6		GT	
		P	N
Pred	P	7115	125
	N	3464	

Test Porcs 0.7		GT	
		P	N
Pred	P	3023	46
	N	7556	

Test Pers. 0.5		GT	
		P	N
Pred	P	1336	3
	N	52	

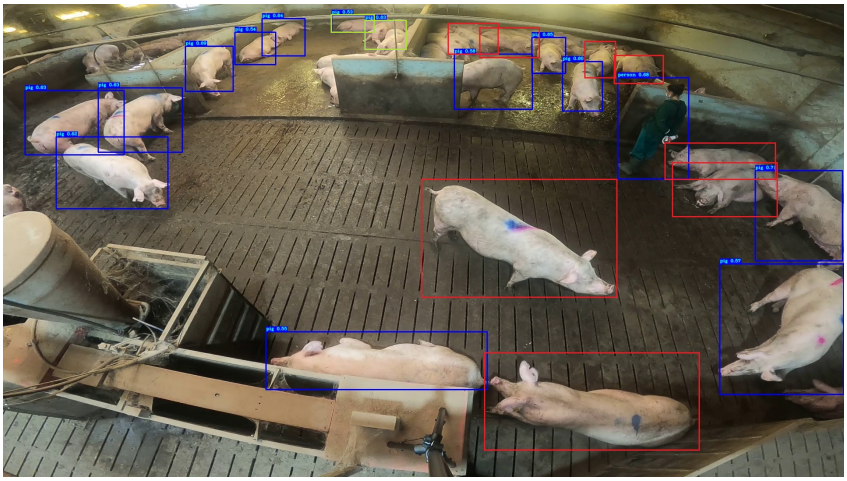
Test Pers. 0.6		GT	
		P	N
Pred	P	1288	0
	N	100	

Test Pers. 0.7		GT	
		P	N
Pred	P	1205	0
	N	183	

Taula 9.2: Resultats deteccions en proves mAP

Com és lògic, el millor resultat dona amb el menor valor d'score threshold, 0.5, on, si s'observa els valors de mAP de validació i test, s'aconsegueix un mAP al voltant del 92%, específicament per als porcs tenim un valor d'un 87% i per a les persones del 96%. A mesura que es va augmentant el score threshold aquests valors disminueixen notòriament, sobre tot afectat pel AP de la predicció dels porcs. Si es miren ara els valors del AP dels porcs de la Universitat d'Edimburg i es comparen amb el de l'IRTA es veu que hi ha molt millor resultat al primer que al segon, on amb un score de 0.5 tenim més del 92% davant al 59% dels de l'IRTA. A mesura que augmenta el score threshold aquesta diferència es manté. Aquesta distància es dona per la diferència de quantitat de dades etiquetades a cada dataset, ja que del dataset de la Universitat d'Edimburg es disposa d'una quantitat notable de dades ja etiquetades per a treballar-hi, mentre que el dataset de l'IRTA s'ha hagut d'etiquetar a mà, cosa que ha reduït notablement la quantitat de dades que es tenen ja que no es disposava del suficient temps com per a generar una quantitat de dades equiparable a l'altre dataset.

a)



b)

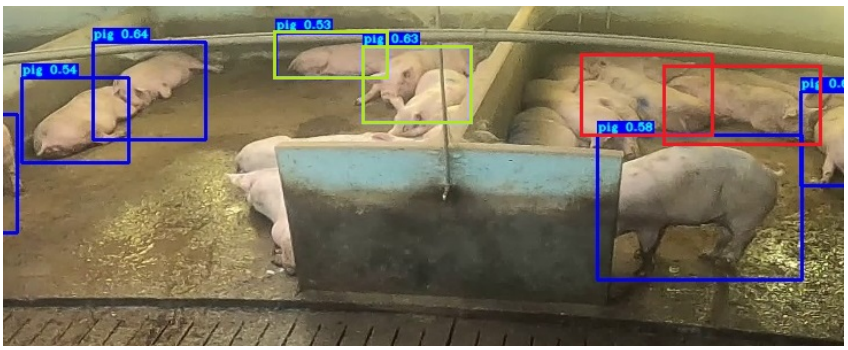


Figura 9.1: Comparació de les deteccions amb el ground truth (GT). En blau les deteccions, en vermell les ROIs del GT no detectades i el verd les deteccions que no són al GT.

Si es fa un anàlisi dels problemes més comuns en la detecció, es troba que els errors més comuns es donen a la figura 9.5:

(a) La persona o un altre porc tapa a l'altre porc el suficient per a que la xarxa no ho detecti.

(b) La persona passa pel davant d'un porc i la xarxa detecta un mateix porc com a dos regions d'interès separades.

(c) Els porcs són animals que tendeixen a estar molt junts, per tant es dona repetidament la situació en que la xarxa confon dos porcs junts com a un mateix animal.

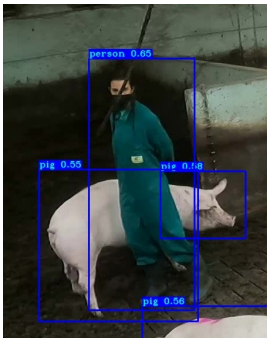
a)  b)  c) 

Figura 9.2: Exemples d'errors comuns a la detecció: a) Persona tapa el porc, b) Detecta un mateix porc com a dos al tenir un obstacle al mig, c) Detectar dos porcs propers com a un sol porc

9.2 Disseny general del sistema

Pel que fa als resultats del sistema general, la combinació de la detecció i el seguiment, no es pot fer un anàlisi quantitatiu dels resultats ja que no es té una mesura per fer-ho, per tant, es farà un anàlisi dels resultats qualitatiu, centrant-se en quins són els problemes principals d'aquest.

Si es torna a parlar dels dos datasets, els resultats obtinguts amb el dataset de la Universitat d'Edimburg són remarcables, aconseguint fer el seguiment de la majoria dels porcs durant seqüències de vídeo completes. A la figura 9.3 es poden observar dos trajectòries de porcs diferents al llarg de la seqüència de vídeo, marcats amb una línia groga i una línia rosa. L'error més donat en les proves d'aquest datasets s'ha donat quan un porc està molt aferrat a un altre, a sobre inclòs, impedint la detecció, seguit del intercanvi del seguiment de dos porcs que es creuen. A la figura 9.4 es pot veure com el porc verd, amb la trajectòria rosa, passa per davant del porc vermell, amb seguiment violeta, i a la imatge c) s'intercanvien els tracks, observant que a la imatge d) el porc verd ha passat a tenir el seguiment lila i el porc vermell el seguiment rosa.

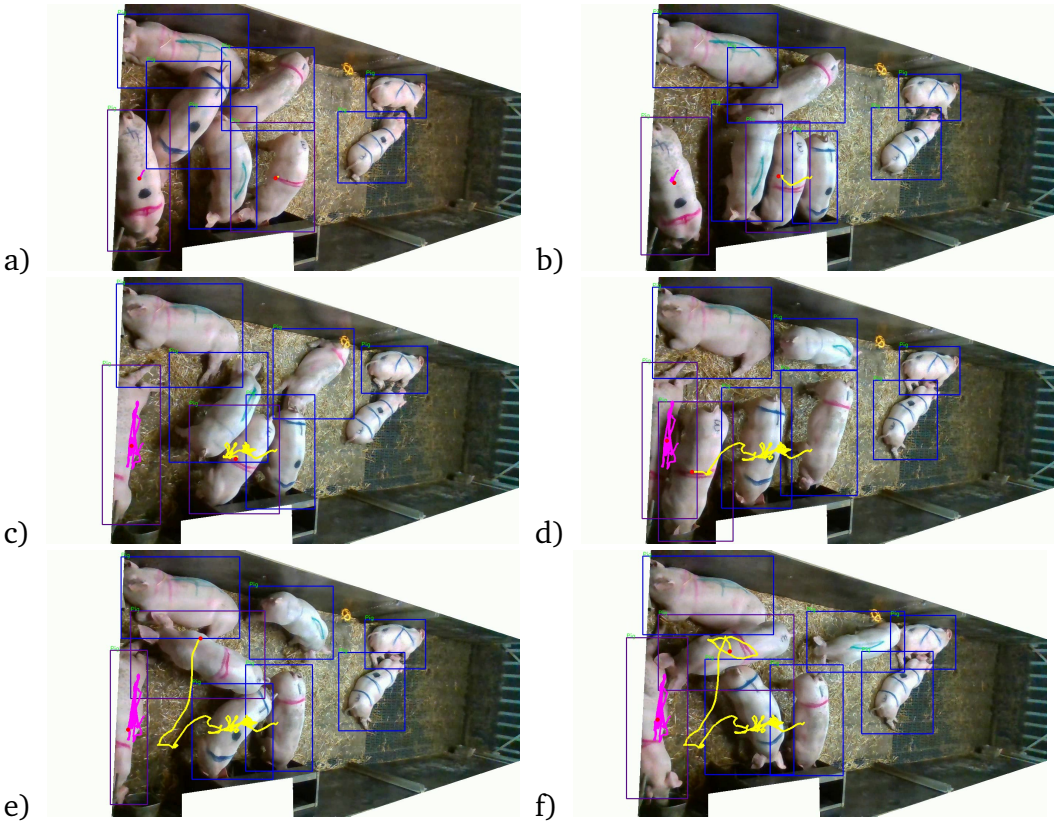


Figura 9.3: Exemple del seguiment de dos porcs.

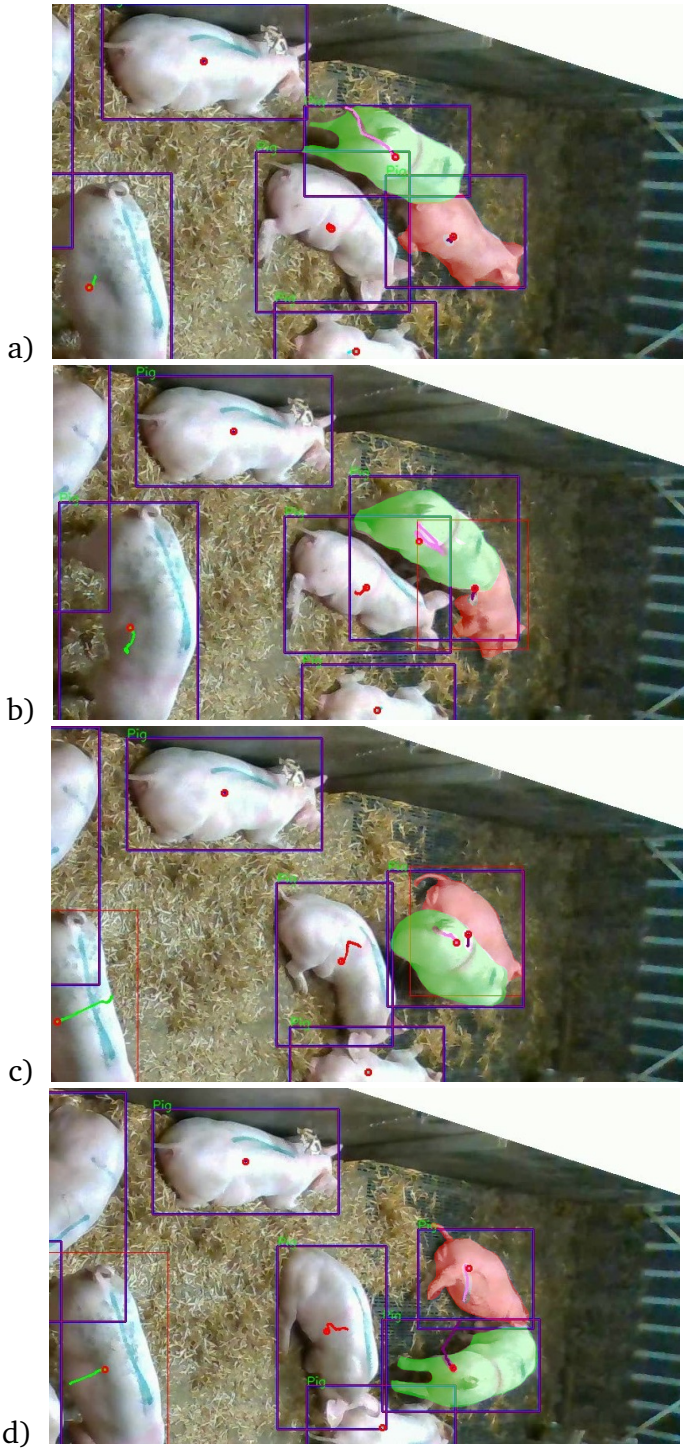


Figura 9.4: Exemple del intercanvi de seguiment entre dos porcs.

Com s'explica als resultats anteriors de la detecció, aquest filtre també serà sensible als errors de la detecció, sobre tot a l'error de la detecció d'un sol animal

com a dos porcs separats, ja que això fa que el filtre crei un altre track que, quan deixa d'haver un obstacle al davant i aquestes dues deteccions passa a ser una, aquest track seguirà existint i no tindrà detecció assignada o, si la té, serà errònia. A la figura 9.5 es pot observar com succeeix això, tenint en compte que les caixes blaves són deteccions de porcs, les verdes de persones i les vermelles la posició predita d'un track la qual no té cap detecció assignada al frame:

(a) Inicialment hi ha una persona caminant en direcció a un porc.

(b) Es veu com la persona comença a tapar al porc i la detecció només capta la part frontal del cos del porc.

(c) S'observa com la xarxa detecta el porc en dues parts ja que la persona obstaculitza la visió.

(d) La persona segueix moguent-se i la xarxa només detecta la part trasera del porc, mentre que el track creat al cap segueix existint, però es mou seguint només la predicció, sense cap mesura.

(e) Es pot veure com només es detecta un porc i el track es va desplaçant cap a la esquerra seguint el moviment que el filtre ha predit que l'objecte fa.

A la figura 9.5 també es pot observar el funcionament del filtre quan hi ha tracks els quals inicialment tenen detecció, seguidament estan algun frame sense detecció i finalment tornen a tenir detecció i recuperen l'objecte i la trajectòria:

(a) Inicialment hi ha una persona caminant detectada per la xarxa, amb una línia rosa que indica la seva trajectòria.

(b) Es veu com la xarxa no detecta la persona i el filtre, amb una caixa vermella, fa la predicció de la trajectòria que ha dut a terme i mou la ROI anterior consequentment i, per tant, la línia rosa mostra també aquesta trajectòria predita.

(c) La xarxa detecta la persona una altra vegada i el filtre la reassigna al track perdut anteriorment.

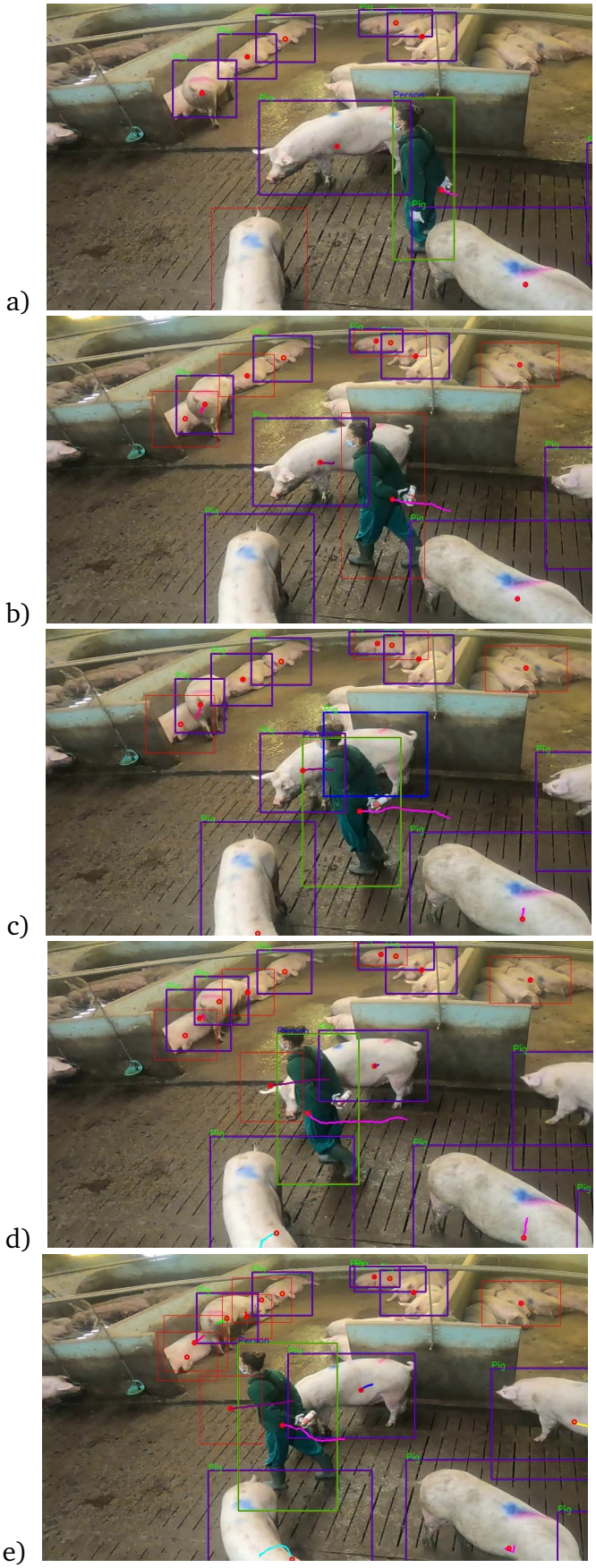


Figura 9.5: Exemple de l'efecte de l'error de detecció doble d'un porc en el seguiment.

CAPÍTOL 10

Conclusions

S'ha vist que a partir de les xarxes neuronals i d'algorismes de seguiment, en aquest cas el Filtre de Kalman en la seva variant Unscended Kalman Filter, és possible dur a terme la detecció i el seguiment de persones i animals com són els porcs i extreure'n informació en temps reduït.

Analitzant els resultats presentats al capítol anterior, la primera conclusió que es treu és la importància que té un bon dataset per a obtenir bons resultats:

- Tenir una quantitat notable de dades.
- Les dades han d'estar ben etiquetades, ja que és la base de l'entrenament.
- En el cas d'aquest projecte, que el punt de vista de les imatges utilitzades sigui l'òptim per a poder extreure'n la màxima quantitat d'informació, en aquest cas en pla zenital.

Un punt que hi ha que remarcar és la diferència notòria de treballar en situacions ideals en comparació a situacions reals. Com s'ha vist, el dataset de la Universitat d'Edimburg es podria entendre aquí com la situació quasi ideal, on el punt de vista és bo, el nombre d'objectes és reduït i les dades ja estan etiquetades. Moltes vegades s'assumeix que aquestes situacions ideals o quasi ideals es donaran sempre i això no és així. Al veure el funcionament del mateix sistema en un entorn real, com són les imatges de l'IRTA, es veu com un sistema que es creia que tenia un funcionament correcte ha de ser modificat per a que pugui funcionar en entorns reals.

Seguidament, es remarca la importància dels paràmetres usats en el desenvolupament de qualsevol algorisme, ja que, com s'ha pogut veure a la prova del Learning Rate, si algun valor és molt gran o molt petit, pot fer que no s'arribi mai al resultat desitjat.

Tot i que els resultats de la detecció estan lluny de ser perfectes, cosa que beneficiaria al sistema de seguiment, aquesta detecció és suficientment bona com per produir resultats de "tracking" acceptables. Des del principi d'aquest projecte es sabia que, per temps i recursos, arribats a aquest punt de la feina uns resultats

com els que es tenen amb un mAP al voltant del 60% per a score thresholds de 0.6 o 0.7 eren els esperats. Com a primer avenç en un projecte d'aquestes magnituds és un bon resultat i obre la porta a poder seguir avançant.

Pel que fa a la viabilitat de continuar amb el projecte, tal i com s'indicava a la introducció, al veure els resultats obtinguts es veu que, tot i que fa falta molt de treball i temps refinament tant dels algorismes com de les dades els quals comportarien la despesa d'una gran quantitat de temps, és una tasca que es podria arribar a dur a terme e implantar en un espai real, tot i que en aquest cas s'haurien d'analitzar factors externs com poden ser la instal·lació del hardware necessari per a dur-ho a terme en l'espai, la il·luminació... Tot i això, en la opinió del alumne, el benefici que aquest sistema podria aportar al benestar dels animals i al benefici econòmic dels ramaders és una bona motivació per a seguir fent feina en aquesta branca de treball.

CAPÍTOL 11

Treball futur

Com ja s'ha dit a la introducció d'aquest document, aquest projecte és part d'un projecte més gran que ha hagut de ser acotat per temps. Pel que fa al treball futur, analitzant els resultats obtinguts i veient on es vol arribar, farà falta dur a terme les següents tasques:

1. Augmentar les dades etiquetades per a poder entrenar millor la xarxa neuronal, ja que un dels majors problemes que hi ha ara és que les deteccions són encara precàries per a que sigui un sistema robust. Es podria seguir el mateix mètode d'etiquetar més imatges del dataset de l'IRTA o intentar aconseguir més dades de tercers.

2. Fer registres dels moviments quantificats per l'algorisme de seguiment i buscar una forma òptima d'emmagatzemar-les i que siguin de fàcil post-precessament.

3. Afegir una xarxa neuronal intermitja entre la YOLOv4 i l'UKF que agafi les deteccions de persones i detecti les parts de cos per a tenir dades més exactes de la relació entre la posició de la persona i el moviment dels animals. Es podria fer ús de la MoveNet o la PoseNet de Tensorflow, per exemple [42].

4. Quan s'hagin dut a terme les tasques anteriors, desenvolupar un algorisme que agafi les dades extretes dels moviments i la pose de les persones i els animals per a fer l'anàlisi de la variació del moviment dels animals quan hi ha persones a prop.

Bibliografia

[1] L. Bergamini, S. Pini, A. Simoni, R. Vezzani, S. Calderara, R. B. Eath, and R. B. Fisher, "Extracting accurate long-term behavior changes from a large pig dataset," in *16th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications, VISIGRAPP 2021*. SciTePress, 2021, pp. 524–533. (Cited on page 1.)

[2] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," *CoRR*, vol. abs/1506.01497, 2015. [Online]. Available: http://arxiv.org/abs/1506.01497 (Cited on pages 3 and 6.)

[3] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," 2020. (Cited on pages 3, 6, 19 and 32.)

[4] K. He, G. Gkioxari, P. Dollar, and R. Girshick, "Mask r-cnn," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017. (Cited on page 3.)

[5] L. Bergamini, S. Pini, A. Simoni, R. Vezzani, S. Calderara, R. Eath, and R. Fisher. (2021) Edinburgh pig behavior video dataset. [Online]. Available: https://homepages.inf.ed.ac.uk/rbf/PIGDATA/ (Cited on pages 4 and 5.)

[6] Conda. (2017) Conda environments. [Online]. Available: https://docs.conda.io/projects/conda/en/latest/user-guide/concepts/environments.html (Cited on page 4.)

[7] L. Bergamini, S. Pini, A. Simoni, R. Vezzani, S. Calderara, R. Eath, and R. Fisher, "Extracting accurate long-term behavior changes from a large pig dataset," in *Proceedings of the 16th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*, G. Farinella, P. Radeva, J. Braz, and K. Bouatouch, Eds., vol. 4. SciTePress, 2021, pp. 524–533, 16th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications, VISIGRAPP 2021 ; Conference date: 08-02-2021 Through 10-02-2021. (Cited on page 5.)

[8] Google. (2018) Open images dataset v4. [Online]. Available: https://storage.googleapis.com/openimages/web/index.html (Cited on pages 5 and 28.)

[9] IRTA. Institut de recerca i tecnologia agroalimentàries. [Online]. Available: https://www.irta.cat/es/ (Cited on page 6.)

[10] AlexeyAB. (2020) darknet. [Online]. Available: https://github.com/AlexeyAB/darknet (Cited on pages 6, 19 and 38.)

[11] P. Flach, *Machine Learning: The Art and Science of Algorithms that Make Sense of Data*. Cambridge University Press, 2012. (Cited on page 11.)

[12] E. Alpaydin, *Introduction to Machine Learning (4th Edition)*. MIT, 2020. (Cited on page 11.)

[13] I. C. Education. (2020) Neural networks. [Online]. Available: https://www.ibm.com/cloud/learn/neural-networks (Cited on page 12.)

[14] R. J. Meinhold and N. D. Singpurwalla, "Understanding the kalman filter," *The American Statistician*, vol. 37, no. 2, pp. 123–127, 1983. (Cited on page 13.)

[15] G. Welch, G. Bishop *et al.*, "An introduction to the kalman filter," 1995. (Cited on page 14.)

[16] S. J. Julier and J. K. Uhlmann, "New extension of the kalman filter to nonlinear systems," in *Signal processing, sensor fusion, and target recognition VI*, vol. 3068. International Society for Optics and Photonics, 1997, pp. 182–193. (Cited on page 15.)

[17] Python Software Foundation. (2021) Python. [Online]. Available: https://www.python.org (Cited on page 17.)

[18] RockyXu66. (2020) Faster rcnn for open images dataset keras. [Online]. Available: https://github.com/RockyXu66/Faster_RCNN_for_Open_Images_Dataset_Keras (Cited on page 17.)

[19] The MathWorks, Inc. (2021) Matlab. [Online]. Available: https://es.mathworks.com (Cited on page 17.)

[20] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2 (Cited on page 18.)

[21] OpenCV team. (2021) Opencv. [Online]. Available: https://opencv.org (Cited on page 18.)

[22] The Matplotlib Development team. (2021) Matplotlib. [Online]. Available: https://matplotlib.org (Cited on page 18.)

[23] NumFOCUS. (2021) Pandas. [Online]. Available: https://pandas.pydata.org (Cited on page 18.)

[24] The scikit-image development team. (2021) scikit-image. [Online]. Available: https://scikit-image.org (Cited on page 18.)

[25] Python Software Foundation. (2021) os. [Online]. Available: https://docs.python.org/3.7/library/os.html?highlight=os#module-os (Cited on page 19.)

[26] ——. (2021) Json. [Online]. Available: https://docs.python.org/3.7/library/json.html (Cited on page 19.)

[27] ——. (2021) random. [Online]. Available: https://docs.python.org/3.7/library/random.html (Cited on page 19.)

[28] ——. (2021) shutil. [Online]. Available: https://docs.python.org/es/3.7/library/shutil.html (Cited on page 19.)

[29] Pythonlessons. (2021) Tensorflow-2.x-yolov3. [Online]. Available: https://github.com/pythonlessons/TensorFlow-2.x-YOLOv3 (Cited on pages 20 and 38.)

[30] Google Brain. (2021) Tensorflow. [Online]. Available: https://www.tensorflow.org (Cited on page 20.)

[31] sj23patel. Object_tracking. [Online]. Available: https://github.com/sj23patel/Object-Tracking (Cited on pages 20 and 43.)

[32] SciPy. Scipy. [Online]. Available: https://scipy.org (Cited on page 20.)

[33] Python Software Foundation. threading. [Online]. Available: https://docs.python.org/3.7/library/threading.html (Cited on page 21.)

[34] ——. copy. [Online]. Available: https://docs.python.org/es/3.7/library/copy.html (Cited on page 21.)

[35] A. Rosebrock. (2016) Intersection over union (iou) for object detection. [Online]. Available: https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/ (Cited on page 30.)

[36] H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. Reid, and S. Savarese, "Generalized intersection over union," June 2019. (Cited on page 30.)

[37] C.-Y. Wang, H.-Y. M. Liao, Y.-H. Wu, P.-Y. Chen, J.-W. Hsieh, and I.-H. Yeh, "Cspnet: A new backbone that can enhance learning capability of cnn," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2020. (Cited on page 32.)

[38] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia, "Path aggregation network for instance segmentation," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8759–8768. (Cited on page 32.)

[39] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 9, pp. 1904–1916, 2015. (Cited on page 32.)

[40] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955. (Cited on page 33.)

[41] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014. (Cited on page 38.)

[42] Tensorflow. Pose estimation. [Online]. Available: https://www.tensorflow.org/lite/examples/pose_estimation/overview (Cited on page 61.)

[43] T. He, Z. Zhang, H. Zhang, Z. Zhang, J. Xie, and M. Li, "Bag of tricks for image classification with convolutional neural networks," *CoRR*, vol. abs/1812.01187, 2018. [Online]. Available: http://arxiv.org/abs/1812.01187 (Cited on page 71.)

Appendices

A.1 Manuals d'usuari

Els següents manuals expliquen com fer ús dels programes per a executar la xarxa neuronal i el sistema de seguiment. Aquest manuals parteixen de la condició de que totes les llibreries, paquets i/o mòduls indicats al capítol 6 estan instal·lats a la màquina que s'utilitza.

A.1.1 Xarxa neuronal

A.1.1.1 Entrenament de la xarxa amb datasets pròpis

1. Inicialment hi ha que modificar el fitxer "configs.py" (Annex C.3.1):

   (a) Comentar les línies 24 i 25 i descomentar les línies 21 i 22 per a definir que es vol fer transfer learning dels pesos del model Darknet, tal com es mostra a la figura 1.

```
1 YOLO_CUSTOM_WEIGHTS          = False
2 YOLO_COCO_CLASSES            = "model_data/coco/coco.names"
3 #YOLO_CUSTOM_WEIGHTS          = True
4 #YOLO_COCO_CLASSES            = "/home/virginia/TFG/YOLOv4/TensorFlow-
  ↪ 2.x-YOLOv3-master/data/annotation.names"
```

Figura 1: Mostra com han de quedar les linies 21-25 del "configs.py"

   (b) Seguidament s'han de configurar els paràmetres de l'entrenament d'aquest mateix fitxer, definits entre les línies 38 i 56. Aquests es poden modificar com sigui més convenient. Els paràmetres que s'han o es poden modificar són:

   - *TRAIN_YOLO_TINY:* Defineix si es vol entrenar un model YOLOv4 Tiny o no. En aquest cas es deixa en False ja que no es treballa amb aquest model.
   - *TRAIN_SAVE_BEST_ONLY:* Defineix si es vol guardar només els pesos del model amb millor resultats en la validació o tots els models entrenats a cada epoch. Es deixa en "True"ja que només interessa el millor resultat i guardar-los tots suposaria un ús més elevat de memòria.
   - *TRAIN_SAVE_CHECKPOINT:* Defineix si es volen guardar tots els millors pesos indicats anteriorment, és a dir, no sobreescriure els

pesos guardats anteriorment. Pel mateix motiu d'abans es deixa en "False".

- *TRAIN_CLASSES:* Indica el path del fitxer on hi ha guardats els noms de les classes sobre les quals entrenar.
- *TRAIN_ANNOT_PATH:* Indica el path de les annotacions o etiquetes a fer ús per entrenar el model.
- *TRAIN_CHECKPOINTS_FOLDER:* Indica el repositori on guardar els pesos del model entrenat.
- *TRAIN_MODEL_NAME:* Indica amb quin nom es guarden els pesos del model entrenat a dintre de la carpeta indicada anteriorment.
- *TRAIN_LOAD_IMAGES_TO_RAM:* Indica si es vol carregar les imatges a la RAM. Això es fa per a processar les imatges més ràpidament a l'hora d'entrenar, tot i que tarda una mica més el carregar el dataset inicialment. Com que el servidor on s'entrena el model a aquest projecte té suficient RAM com per carregar totes les imatges, aquí està a "True", però dependrà de la capacitat de la que es disposi.
- *TRAIN_BATCH_SIZE:* Indica quantes imatges es processen a cada iteració de cada epoch. En aquest cas, com que hi ha un nombre considerable de imatges a processar s'ha elegit fer-ho de quatre en quatre després d'algunes proves, però aquest nombre és variable.
- *TRAIN_DATA_AUG:* Indica si es vol realitzar "Data Augmentation"que bàsicament és aplicar algunes transformacions a imatges ja existents del dataset per a tenir més dades que processar.
- *TRAIN_TRANSFER:* Indica si es vol entrenar la xarxa des d'un altre model ja definit, com el Darknet en aquest cas, o si es vol entrenar de zero.
- *TRAIN_FROM_CHECKPOINT:* Indica si es vol començar a entrenar des d'un altre checkpoint, és a dir, un entrenament anterior. En el cas d'aquest projecte sempre s'ha treballat amb els pesos inicials del Darknet, mai d'entrenaments anteriors.
- *TRAIN_LR_INIT:* Indica el learning rate que es vol tenir al llarg del entrenament.
- *TRAIN_LR_END:* Indica el learning rate inicials que es vol seguir als epochs de calentament o warm up. Els epochs de calentaments són epochs duts a terme al inici del entrenament per a pujar progressivament d'un learning rate petit (en aquest cas el

TRAIN_LR_END) a un learning rate més gran que es vol mantenir al llarg del entrenament (en aquest cas el TRAIN_LR_INIT). Això es fa ja que al inici del entrenament normalment els paràmetres usats tenen valors aleatoris que disten molt de la solució esperada i fer ús d'un learning rate gran pot fer que hi hagi inestabilitat numèrica [43].

- *TRAIN_WARMUP_EPOCHS:* Indica quants d'epochs de calentament es volen fer.
- *TRAIN_EPOCHS:* Indica quants d'epochs d'entrenament es volen fer.

A la figura 2 es mostra la configuració del últim entrenament dut a terme en aquest projecte.

```
# Train options
TRAIN_YOLO_TINY             = False
TRAIN_SAVE_BEST_ONLY        = True # saves only best model according
 ↪  validation loss (True recommended)
TRAIN_SAVE_CHECKPOINT       = False # saves all best validated
 ↪  checkpoints in training process (may require a lot disk space)
 ↪  (False recommended)
TRAIN_CLASSES               = "/home/virginia/TFG/YOLOv4/TensorFlow-2
 ↪  .x-YOLOv3-master/data/annotation.names"
TRAIN_ANNOT_PATH            = "/home/virginia/TFG/YOLOv4/TensorFlow-2
 ↪  .x-YOLOv3-master/data/PigPersv2/annotation_train.txt"
TRAIN_LOGDIR                = "log"
TRAIN_CHECKPOINTS_FOLDER    = "checkpoints/yolov4_PigPersv5"
TRAIN_MODEL_NAME            = f"{YOLO_TYPE}_custom"
TRAIN_LOAD_IMAGES_TO_RAM    = True # With True faster training, but
 ↪  need more RAM
TRAIN_BATCH_SIZE            = 4
TRAIN_INPUT_SIZE            = 416
TRAIN_DATA_AUG              = True
TRAIN_TRANSFER              = True
TRAIN_FROM_CHECKPOINT       = False # "checkpoints/yolov3_custom"
TRAIN_LR_INIT               = 5e-6
TRAIN_LR_END                = 5e-7
TRAIN_WARMUP_EPOCHS         = 1
TRAIN_EPOCHS                = 55
```

Figura 2: Mostra com ha de quedar la configuració de l'entrenament del "configs.py"

(c) Les últimes configuracions a fer en aquest fitxer són els paràmetres de validació o test, entre les línies 58 i 65 del fitxer. En aquest cas hi ha els següents paràmetres modificables:

- *TEST_ANNOT_PATH:* Equival al TRAIN_ANNOT_PATH però amb el fitxer de validació.
- *TEST_BATCH_SIZE:* Equival al TRAIN_BATCH_SIZE però per a la validació dels resultats.
- *TEST_DATA_AUG:* Equival al TRAIN_DATA_AUG però per a la validació dels resultats.
- *TEST_SCORE_THRESHOLD:* És el valor mínim de la confiança que té la xarxa en el resultat, és a dir, la probabilitat que té una regió d'interès de ser el que la xarxa ha predit, que s'accepta.
- *TEST_IOU_THRESHOLD:* És el valor mínim que han de tenir dos regions d'interès d'IoU per a acceptar-les com a una sola regió d'interès o objecte.

A la figura 3 es mostren aquests paràmetres.

```
# TEST options
TEST_ANNOT_PATH               = "/home/virginia/TFG/YOLOv4/TensorFlow-2.x-YOLOv3-master/data/PigPersv2/annotation_val.txt"
TEST_BATCH_SIZE               = 4
TEST_INPUT_SIZE               = 416
TEST_DATA_AUG                 = False
TEST_DECTECTED_IMAGE_PATH     = ""
TEST_SCORE_THRESHOLD          = 0.7
TEST_IOU_THRESHOLD            = 0.5
```

Figura 3: Mostra com ha de quedar la configuració de la validació o test del "configs.py"

2. Seguidament s'ha de modificar la línia 42 del fitxer "train.py"(Annex C.3.5) i indicar el fitxer de text on es vol guardar el procés d'entrenament de la xarxa.

```
# Create register file for training process (Virginia Ramón)
file = open("loss_training_PigPersv5.txt", 'w')
```

Figura 4: Línia a modificar del fitxer "train.py"

3. Quan ja s'han fet les modificacions pertinents, hi ha dos opcions per a executar el programa:

    - Si es té algun programa o aplicació on es pugui compilar i executar un programa de Python directament, executar el fitxer "train.py".
    - Si no es disposa de tal interficie, obrir un terminal a la màquina, anar al repositori on hi ha guardats els fitxers de codi de la YOLOv4 i executar la comanta "*python train.py*".

4. Si tot va bé, es mostrarà per pantalla la informació del Tensorflow (figura 5), seguidament del progres de càrrega dels datasets (figura 6) i finalment el progrés de cada epoch (figura 7).

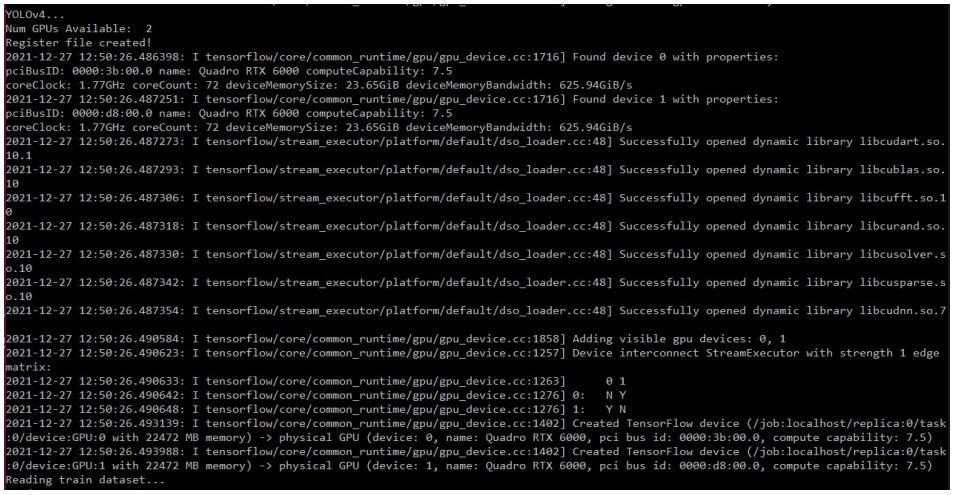


Figura 5: Sortida inicialització del Tensorflow

Figura 6: Sortida progrés de càrrega dels datasets

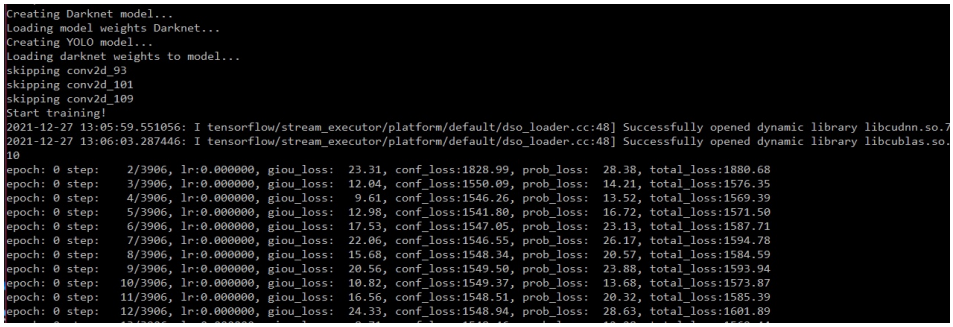


Figura 7: Sortida del entrenament de la xarxa

A.1.1.2 Comprovació mAP d'un model entrenat

1. Inicialment hi ha que modificar el fitxer "configs.py" (Annex C.3.1):

   (a) Comentar les línies 21 i 22 i descomentar les línies 24 i 25 per a definir que es vol fer ús del model entrenat, tal com es mostra a la figura 8. Substituir el paràmetre YOLO_COCO_CLASSES pel fitxer tipus text de les classes.

```
1 #YOLO_CUSTOM_WEIGHTS          = False
2 #YOLO_COCO_CLASSES            = "model_data/coco/coco.names"
3 YOLO_CUSTOM_WEIGHTS          = True
4 YOLO_COCO_CLASSES            = "/home/virginia/TFG/YOLOv4/TensorFlow-2
  ↪ .x-YOLOv3-master/data/annotation.names"
```

Figura 8: Mostra com han de quedar les linies 21-25 del "configs.py"

   (b) Cal indicar el dataset amb el que es vol fer la prova de mAP i els paràmetres de testing. Per fer això cal modificar les línies 58 a 65 del fitxer amb les dades que es volen. A la figura 9 es mostren aquests paràmetres.

```
# TEST options
TEST_ANNOT_PATH               = "/home/virginia/TFG/YOLOv4/TensorFlow-2.x-YOLOv3-master/data/PigPersv2/test.txt"
TEST_BATCH_SIZE               = 4
TEST_INPUT_SIZE               = 416
TEST_DATA_AUG                 = False
TEST_DECTECTED_IMAGE_PATH     = ""
TEST_SCORE_THRESHOLD          = 0.5
TEST_IOU_THRESHOLD            = 0.5
```

Figura 9: Mostra com ha de quedar la configuració de test del "configs.py"

2. Quan ja s'han fet les modificacions pertinents, hi ha dos opcions per a executar el programa:

   - Si es té algun programa o aplicació on es pugui compilar i executar un programa de Python directament, executar el fitxer "mAP-test.py"(Annex C.3.8).
   - Si no es disposa de tal interficie, obrir un terminal a la màquina, anar al repositori on hi ha guardats els fitxers de codi de la YOLOv4 i executar la comanta "*python mAPtest.py*" (Annex C.3.8).

3. Si tot va bé, es mostrarà per pantalla la informació del Tensorflow (figura 10), seguidament del progres de càrrega dels datasets (figura 11) i finalment el resultat dels APs i el mAP (figura 12).

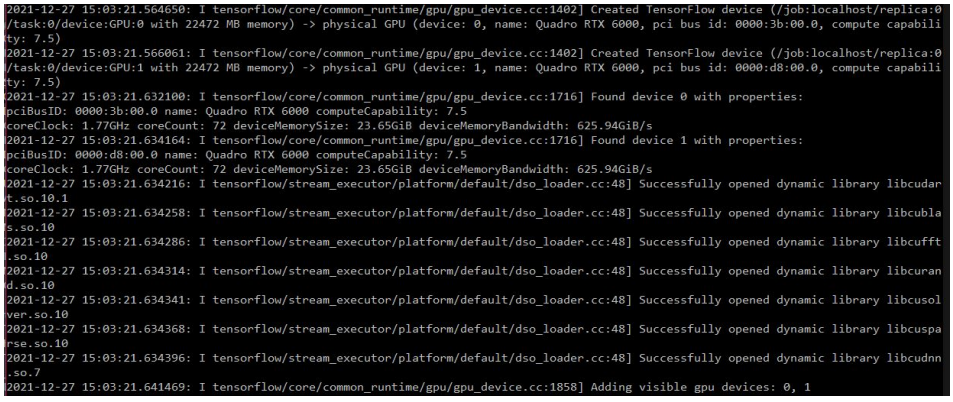


Figura 10: Sortida inicialització del Tensorflow

Figura 11: Sortida progrés de càrrega dels datasets

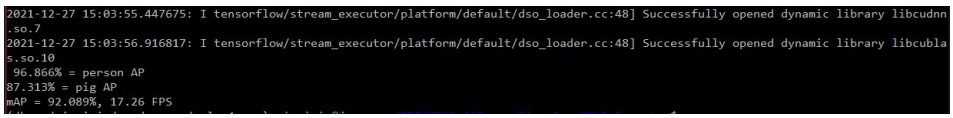


Figura 12: Sortida del resultat dels APs i mAP

A.1.1.3 Predicció

Per a fer proves amb imatges i seqüències de vídeo hi ha que seguir els següents passos:

1. Inicialment hi ha que modificar el fitxer ”configs.py” (Annex C.3.1), comentar les línies 21 i 22 i descomentar les línies 24 i 25 per a definir que es vol fer ús del model entrenat, tal com es mostra a la figura 13. Substituir el paràmetre YOLO_COCO_CLASSES pel fitxer tipus text de les classes.

```
1 #YOLO_CUSTOM_WEIGHTS          = False
2 #YOLO_COCO_CLASSES            = "model_data/coco/coco.names"
3 YOLO_CUSTOM_WEIGHTS          = True
4 YOLO_COCO_CLASSES            = "/home/virginia/TFG/YOLOv4/TensorFlow-2
  ↪ .x-YOLOv3-master/data/annotation.names"
```

Figura 13: Mostra com han de quedar les linies 21-25 del ”configs.py”

2. Quan ja s'ha modificat el fitxer anterior, s'ha de modificar el fitxer ”detection_demo.py” (Annex C.3.6):
   (a) Primer s'ha d'indicar si es vol fer la detecció sobre una imatge o una seqüència de vídeo modificant la línia 23 indicant ”True” si es vol una imatge, ”False” altrament.
   (b) Si es vol usar una imatge, hi ha que modificar les línies:
   - *Línia 30:* Indicar el path de la imatge sobre la que fer la detecció.
   - *Línia 31:* Indicar el path de la imatge resultant de la detecció.

- *Línia 32:* Es poden modificar els paràmetres "show"que indica si es vol mostrar per pantalla la imatge resultant, "score_thresholdï el color del rectangle per a indicar les regions d'interés amb "rectangle_colors".
- *Línia 43:* Indica el fitxer on guardar el resultat de la detecció.

A la figura 15 es mostra com ha de quedar el codi.

```
1  image_path   = "/home/virginia/TFG/PIGDATA/annotated/frames/2019_11_1
   ↪  1_000028_146.jpg"
2  image_out_path = "./IMAGES/pigs_pred.jpg"
3  _, data, classes = detect_image(yolo, image_path, image_out_path,
   ↪  input_size=YOLO_INPUT_SIZE, show=False, score_threshold=0.5,
   ↪  rectangle_colors=(255,0,0))
4
5  # Data to txt
6  datafile = []
7  fr = str(data[0])
8  bbxs = ""
9  for bb in data[1]:
10     pos = str(bb[0])+","+str(bb[1])+","+str(bb[2])+","+str(bb[3])+","
       ↪  +str(bb[5])
11     bbxs= bbxs + " " + pos
12 datafile = fr+" "+bbxs+"\n"
13
14 file = open("./IMAGES/pred_img.txt", 'w')
15 file.write(datafile)
16 file.close()
17 print("Image detection ended")
```

Figura 14: Mostra com han de quedar les linies 30-46 del "detection_demo.py"

(c) Si es vol usar una imatge, hi ha que modificar les línies:

- *Línia 49:* Indicar el path del video sobre el que fer la detecció.
- *Línia 50:* Indicar el path de la vídeo resultant de la detecció.
- *Línia 51:* Es poden modificar els paràmetres "show"que indica si es vol mostrar per pantalla la imatge resultant, "score_thresholdï el color del rectangle per a indicar les regions d'interés amb "rectangle_colors".
- *Línia 63:* Indica el fitxer on guardar el resultat de la detecció.

A la figura 15 es mostra com ha de quedar el codi.

```
video_path   = "./IMAGES/GH010013_zenital_Test.mp4"
video_out_path = "./IMAGES/GH010013_zenital_Test_pred.mp4"
data, classes = detect_video(yolo, video_path, video_out_path,
    input_size=YOLO_INPUT_SIZE, show=False, score_threshold=0.5,
    rectangle_colors=(255,0,0))

# Data to txt
datafile = []
for frame in data:
    fr = str(frame[1])
    bbxs = ""
    for bb in frame[2]:
        pos = str(bb[0])+","+str(bb[1])+","+str(bb[2])+","+str(bb[3])
            +","+str(bb[5])
        bbxs= bbxs + " " + pos
    datafile.append(fr+" "+bbxs)

file = open("./IMAGES/pred_video.txt", 'w')
for d in datafile:
   file.write(d+"\n")
file.close()
print("Video detection ended")
```

Figura 15: Mostra com han de quedar les linies 49-67 del ”detection_demo.py”

3. Quan ja s'han fet les modificacions pertinents, hi ha dos opcions per a executar el programa:
   - Si es té algun programa o aplicació on es pugui compilar i executar un programa de Python directament, executar el fitxer ”detection_demo.py”.
   - Si no es disposa de tal interficie, obrir un terminal a la màquina, anar al repositori on hi ha guardats els fitxers de codi de la YOLOv4 i executar la comanta "*python detection_demo.py*".
4. Si tot va bé, es mostrarà per pantalla la informació del Tensorflow (figura 16). Si és una imatge, seguidament es mostrarà un missatge de ïmage detection ended"si tot ha ha anat bé (figura 17). En canvi, si és un vídeo, es mostrarà el progrés de les deteccions (figura 18) i finalment un missatge de "Video detection ended"(figura 19).

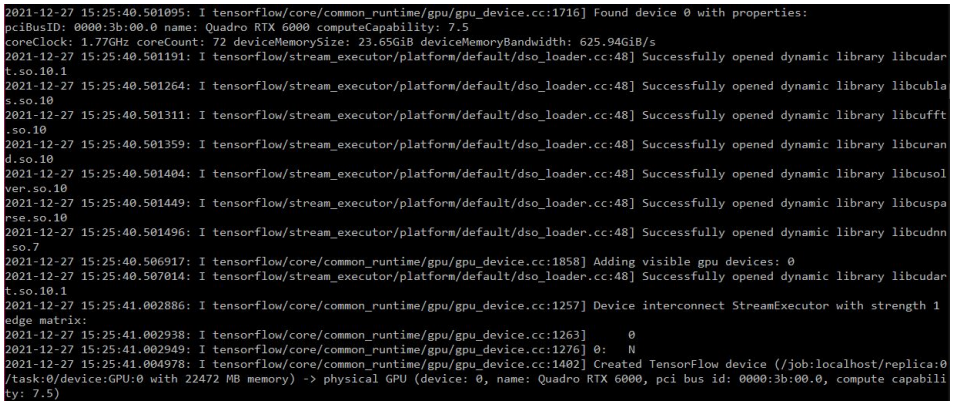


Figura 16: Sortida inicialització del Tensorflow

Figura 17: Missatge final detecció d'imatge

Figura 18: Progrés de la detecció de vídeo

Figura 19: Missatge final detecció de vídeo

A.1.2 Ús del sistema de seguiment amb prediccions de la xarxa neuronal

Per a poder fer ús del sistema de seguiment amb les prediccions del model entrenat de la YOLOv4 hi ha que seguir els següents passos:

1. Inicialment hi ha que modificar les següents línies del fitxer "object_tracking.py" (Annex D.4.3):

   (a) *Línia 63:* Indicar en la captura de vídeo el path del fitxer de vídeo a fer el seguiment.

(b) *Línia 65:* Indicar el path on guardar el vídeo resultant d'aplicar la detecció i l'algorisme.

(c) *Línia 78:* Definir els paràmetres de la funció Tracker:

- *1r paràmetre:* Defineix l'IoU mínim entre una detecció i l'última regió d'interès d'un dels seguiments per a tenir-lo en compte com a possible detecció a afegir a aquest seguiment.
- *2n paràmetre:* Indica el nombre de frames que es manté un seguiment sense detecció abans de esborrar-lo de la llista.
- *3r paràmetre:* Indica el nombre màxim de frames en que es pot mantenir un seguiment, indiferentment de si hi ha detecció o no.
- *4t paràmetre:* Indica la identificació de cada objecte de seguiment.

A la figura 20 es mostra com ha de quedar el codi.

```
1  # Create opencv video capture object
2  cap = cv2.VideoCapture("../IMAGES/GH010020_Trim.mp4")
3  #Output path to save video result
4  output_path = "../IMAGES/GH010020_Trim_UKF.mp4"
5
6  cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1920)
7  cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 1080)
8
9  # by default VideoCapture returns float instead of int
10 width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
11 height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
12 fps = int(cap.get(cv2.CAP_PROP_FPS))
13 codec = cv2.VideoWriter_fourcc(*'XVID')
14 out = cv2.VideoWriter(output_path, codec, fps, (width, height)) #
   ↪  output_path must be .mp4
15
16 # Create Object Tracker
17 tracker = Tracker(0.4, 80, 10000, 20)
```

Figura 20: Mostra com han de quedar les linies 62-78 del "object_tracking.py"

2. Quan ja s'ha configurat el fitxer anterior, s'ha de configurar la detecció d'objectes amb el model entrenat de la YOLOv4. Inicialment hi ha que modificar el fitxer "configs.py" (Annex C.3.1), comentar les línies 21 i 22

i descomentar les línies 24 i 25 per a definir que es vol fer ús del model entrenat, tal com es mostra a la figura 21. Substituir el paràmetre YOLO_COCO_CLASSES pel fitxer tipus text de les classes.

```
#YOLO_CUSTOM_WEIGHTS          = False
#YOLO_COCO_CLASSES            = "model_data/coco/coco.names"
YOLO_CUSTOM_WEIGHTS          = True
YOLO_COCO_CLASSES            = "/home/virginia/TFG/YOLOv4/TensorFlow-2
↪ .x-YOLOv3-master/data/annotation.names"
```

Figura 21: Mostra com han de quedar les linies 21-25 del "configs.py"

3. Seguidament hi ha que modificar la línia 21 del fitxer "frame_detection.py"(Annex D.4.5) per a indicar els paràmetres de la funció de detecció d'objectes. Aquí es pot modificar el valor del "score_thresholdï del ïou_thresholdüsat a cada detecció. A la figura 22 es mostra com ha de quedar el codi.

```
            # Predict Bboxes
            bboxes, classes = detect_frame(net, frame,
            ↪ input_size=YOLO_INPUT_SIZE, show=False,
            ↪ score_threshold=0.5, iou_threshold=0.5)
```

Figura 22: Mostra com han de quedar les línies 20-21 del "frame_detection.py"

4. Quan ja s'han fet les modificacions pertinents, hi ha dos opcions per a executar el programa:

   - Si es té algun programa o aplicació on es pugui compilar i executar un programa de Python directament, executar el fitxer "object_tracking.py".
   - Si no es disposa de tal interficie, obrir un terminal a la màquina, anar al repositori on hi ha guardats els fitxers de codi de la YOLOv4 i executar la comanta "*python object_tracking.py*".

5. Si tot va bé, es mostrarà per pantalla la informació del Tensorflow (figura 23). Si no hi ha errors, es mostrarà per pantalla l'anàlisi de frame a frame on es mostrarà el nombre de objectes de seguiment, la velocitat de detecció en FPS, el temps que ha tardat, el nombre de deteccions (BBXS), l'assignació de deteccions als objectes de seguiment i les deteccions que

no han pogut ser assignades i per tant creat un nou objecte de seguiment (figura 24) i finalment un missatge de "video ended"(figura 24).

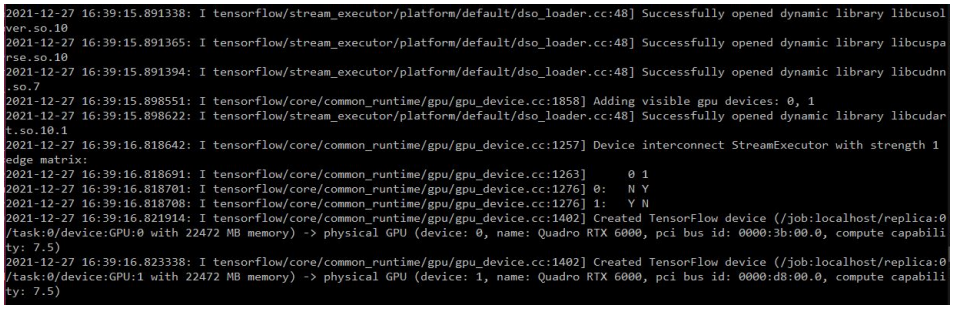


Figura 23: Sortida inicialització del Tensorflow

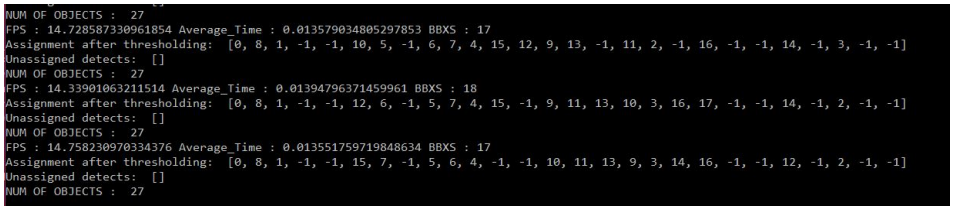


Figura 24: Sortida del progrés del algorisme frame a frame

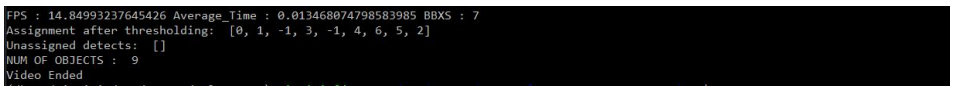


Figura 25: Missatge final del processament del algorisme

B.2 Codi Datasets

B.2.1 extractFrameData.py

```
import numpy as np
import cv2
import os
import json

# Define path for box information folder, video folder and mask file
trackPath = '/home/virginia/TFG/PIGDATA/annotated'
maskPath = "/home/virginia/TFG/PIGDATA/mask.jpg"

Folders = [x[0] for x in os.walk(trackPath)]
Folders.pop(0)

# Partial path for images
partialPath = trackPath + "/frames"

frames = False

video = 1
for folder in Folders:
    folderName = os.path.basename(folder)
    # Extract subdirectories for box information
        # Each subdirectory name in trackPath equals a subdirecroty name in videoPath
    subFolders = [x[0] for x in os.walk(folder)]
    subFolders.pop(0)
    #To select only first 30 folders
    #subFolders = sorted(subFolders)
    #subFolders = subFolders[0:29]
    for f in subFolders:
        print(f)
        if frames:
            #Read mask image
            mask = cv2.bitwise_not(cv2.imread(maskPath, cv2.IMREAD_COLOR))

            # Indicates the frame number to save frame
```

```
35              idx = 0
36
37              # Extract folder name
38              subfolderName = os.path.basename(f)
39
40              print(str(video) + " -> " + subfolderName)
41              video += 1
42
43              # Create a VideoCapture object and read video
                 ↪  of folder
44              cap = cv2.VideoCapture(f+"/color.mp4")
45
46              # Check if camera opened successfully
47              if (cap.isOpened()== False):
48                  print("Error opening video stream or file")
49
50              file_path = partialPath+'/'+folderName+'_'+subf⌋
                 ↪  olderName+"_"
51
52              # Read until video is completed
53              while(cap.isOpened()):
54              # Capture frame-by-frame
55                  ret, frame = cap.read()
56                  if ret == True:
57                      # Add mask and frame (to erase the
                         ↪  outside of the image)
58                          newFrame = cv2.add(frame, mask)
59                      # Save new frame as single image
60                          cv2.imwrite(file_path+str(idx)+".j⌋
                             ↪  pg",
                             ↪  newFrame)
61                          idx += 1
62                  # Break the loop
63                  else:
64                      break
65
66              # When everything done, release the video
                 ↪  capture object
67              cap.release()
68          else:
69              # Extract info BBs
```

```
70              # track[0] --> frame rate (FPS) while
                ↪ generating tracklets of a single video
71              # track[1] --> mean tracklet length
72              # track[2] --> video output resolution (should
                ↪ be always 1280x720 pixels)
73              # track[3] --> N-dimensional (N = number of
                ↪ tracklets) list. Each list entry contains
                ↪ an M-dimensional (M = tracklet length)
                ↪ array representing a single tracklet.
74              #               Each entry in the tracklet
                ↪ array has 6 values [frameID, bbox_X_min,
                ↪ bbox_Y_min, bbox_X_max, bbox_Y_max,
                ↪ tracklet_state],
75              #               "tracklet_state" can have 3
                ↪ values: 0 = new tracklet, 1 = updated
                ↪ tracklet, 2 = wrong tracklet or no
                ↪ available detection
76
77              # Open file to write annotations on
78              # Extract folder name
79              #print(f)
80              subfolderName = os.path.basename(f)
81              file = open(trackPath+ '/annotations' +
                ↪ "/annotation_"+folderName+'_'+subfolderName
                ↪ +".txt",
                ↪ 'w')
82
83              data = []
84
85              # Read tracklets.py
86              track = trackPath+'/'+folderName+'/'+subfolderN
                ↪ ame+'/output.json'
87              with open(track) as f:
88                  dict = json.load(f)
89              obj = np.array(dict["objects"])
90              for o in obj:
91                  last_fr = None
92                  # Extract BB information
93                  track = np.array(o["frames"])
94                  #Define class name of objects
95                  class_name = "Pig"
```

```
                ap = False
                count = 0
                # For each tracklet saved
                for t in track:
                    # If tracklet discontinued
                    while ap and t['frameNumber'] > count:
                        file_path = partialPath+'/'+folderName+'_'+subfolderName+"_"+str(count*3)+".jpg"
                        saved = False
                        if data != []:
                            for fr in data:
                                if fr[0] == file_path:
                                    fr[2].append(str(last_fr['bbox']['x'])+","+str(last_fr['bbox']['y'])+","+str(last_fr['bbox']['x']+last_fr['bbox']['width'])+","+str(last_fr['bbox']['y']+last_fr['bbox']['height'])+",0")
                                    saved = True
                                    break
                        if not saved:
                            data.append([file_path,3*count,[str(last_fr['bbox']['x'])+","+str(last_fr['bbox']['y'])+","+str(last_fr['bbox']['x']+last_fr['bbox']['width'])+","+str(last_fr['bbox']['y']+last_fr['bbox']['height'])+",0"]])
                        count+=1
                    # If tracklet actual
                    file_path = partialPath+'/'+folderName+'_'+subfolderName+"_"+str(3*t['frameNumber'])+".jpg"
                    saved = False
                    if data != []:
```

```
117                         for fr in data:
118                             if fr[0] == file_path:
119                                 fr[2].append(str(t['bbox'][
                                     ↪ 'x'])+","+str(t['bbox']
                                     ↪ ['y'])+","+str(t['bbox'
                                     ↪ ]['x']+t['bbox']['width
                                     ↪ '])+","+str(t['bbox']['
                                     ↪ y']+t['bbox']['height']
                                     ↪ )+",0")
120                                 saved = True
121                                 break
122                     if not saved:
123                         data.append([file_path,3*t['frameN
                             ↪ umber'],[str(t['bbox']['x'])+",
                             ↪ "+str(t['bbox']['y'])+","+str(t
                             ↪ ['bbox']['x']+t['bbox']['width'
                             ↪ ])+","+str(t['bbox']['y']+t['bb
                             ↪ ox']['height'])+",0"]])
124                     last_fr = t
125                     ap = True
126                     count = t['frameNumber']+1
127                 # If tracklet not finished
128                 while 3*count<1800:
129                     file_path =
                         ↪ partialPath+'/'+folderName+'_'+subf
                         ↪ olderName+"_"+str(count*3)+".jpg"
130                     saved = False
131                     if data != []:
132                         for fr in data:
133                             if fr[0] == file_path:
134                                 fr[2].append(str(last_fr['
                                     ↪ bbox']['x'])+","+str(la
                                     ↪ st_fr['bbox']['y'])+","
                                     ↪ +str(last_fr['bbox']['x
                                     ↪ ']+last_fr['bbox']['wid
                                     ↪ th'])+","+str(last_fr['
                                     ↪ bbox']['y']+last_fr['bb
                                     ↪ ox']['height'])+",0")
135                                 saved = True
136                                 break
137                     if not saved:
```

```
                    data.append([file_path,3*count,[str(last_fr['bbox']['x'])+","+str(last_fr['bbox']['y'])+","+str(last_fr['bbox']['x']+last_fr['bbox']['width'])+","+str(last_fr['bbox']['y']+last_fr['bbox']['height'])+",0"]])
                count+=1

            # Define text to annotate in .txt file in format (file_path,x1,y1,x2,y2,class_name)
            data.sort(key=lambda x: x[1])
            #print(data)
            #Write lines in annotation file
            for d in data:
                text = d[0]
                for f in d[2]:
                    text = text + " " + f
                file.write(text+"\n")
            file.close()
```

B.2.2 Extract_frames.py

```
import numpy as np
import cv2

video = ("./")
# Create a VideoCapture object and read video of folder
cap = cv2.VideoCapture("./GH010008_Test.mp4")

# Check if camera opened successfully
if (cap.isOpened()== False):
    print("Error opening video stream or file")

i = 0

# Read until video is completed
while(cap.isOpened()):
# Capture frame-by-frame
    ret, frame = cap.read()
    if ret == True:
        if i%10 == 0:
        # Save new frame as single image
            cv2.imwrite("./frames_persona/GH010008_Test_Pe
            ↪ rs_" +str(i)+ ".jpg",
            ↪ frame)
        i += 1
    # Break the loop
    #else:
    #    break

# When everything done, release the video capture object
cap.release()
```

B.2.3 breafing_datasets.py

```
import os
import sys
import cv2
import random
import numpy as np

# Check how many pigs and people are in a dataset
def load_annotations(annot_path):
    count_pig = 0
    count_pers = 0
    final_annotations = []
    with open(annot_path, 'r') as f:
        txt = f.read().splitlines()
        annotations = [line.strip() for line in txt if
         ↪  len(line.strip().split()[1:]) != 0]
    n = 1

    for annotation in annotations:
        # fully parse annotations
        line = annotation.split(" ")
        del line[0]

        for l in line:
            x = l.split(',')

            if x[4] == '0':
                count_pig+=1
            else:
                count_pers+=1

    return count_pig, count_pers

count_pig, count_pers = load_annotations("./data/PigPersv2
 ↪  /annotation_train.txt")
print("Train")
print(count_pig)
print(count_pers)

```

```
count_pig, count_pers =
    load_annotations("./data/PigPersv2/annotation_val.txt")
print("Val")
print(count_pig)
print(count_pers)

count_pig, count_pers = load_annotations("./data/PigPersv2
    /annotation_test.txt")
print("Test")
print(count_pig)
print(count_pers)
```

B.2.4 ExtractROIs.m

```
clear all
close all

MyFolderInfo = dir('frames/GH010013_zenital_Train*');
i = 1;
rois =[];
global KEY_IS_PRESSED;
for f = 221:240
    disp(f);
    rois_img = [];
    string = strcat('./frames/',MyFolderInfo(f).name);
    I = imread(string); imshow(I);
    KEY_IS_PRESSED = 0;
    gcf;
    % Select all pigs
    set(gcf, 'KeyPressFcn', @myKeyPressFcn);
    while ~KEY_IS_PRESSED
          roi = drawrectangle();
          if ~KEY_IS_PRESSED
            disp("pig");
            rois_img = [rois_img ;[roi.Position]];
          end
    end
    rois_img2 = [];
    disp("next");
    KEY_IS_PRESSED = 0;
    gcf;
    % Select all people
    set(gcf, 'KeyPressFcn', @myKeyPressFcn);
    while ~KEY_IS_PRESSED
          roi = drawrectangle();
          if ~KEY_IS_PRESSED
            disp("person");
            rois_img2 = [rois_img2 ; [roi.Position]];
          end
    end
    % Save data
    rois_p.name = string;
    rois_p.bboxes_pigs = rois_img;
```

```
40      rois_p.bboxes_pers = rois_img2;
41      rois = [rois rois_p];
42      i = i+1;
43  end
44
45  % Save BBxs data
46  fileID =
    ↪  fopen('annotations_GH010013_zenital_Train_221to240.txt','w');
47  for f = 1:size(rois,2)
48      text = rois(f).name;
49      % Save pigs BBxs
50      for i = 1:size(rois(f).bboxes_pigs,1)
51          text = strcat(text,"
    ↪  ",int2str(rois(f).bboxes_pigs(i,1)),
    ↪  ",",int2str(rois(f).bboxes_pigs(i,2)),",",
    ↪  int2str(rois(f).bboxes_pigs(i,1)+rois(f).bboxes_pigs(i,3)),",",
    ↪  int2str(rois(f).bboxes_pigs(i,2)+rois(f).bboxes_pigs(i,4)),",0");
52      end
53      % Save people BBxs
54      for i = 1:size(rois(f).bboxes_pers,1)
55          text = strcat(text,"
    ↪  ",int2str(rois(f).bboxes_pers(i,1)),
    ↪  ",",int2str(rois(f).bboxes_pers(i,2)),",",
    ↪  int2str(rois(f).bboxes_pers(i,1)+rois(f).bboxes_pers(i,3)),",",
    ↪  int2str(rois(f).bboxes_pers(i,2)+rois(f).bboxes_pers(i,4)),",1");
56      end
57      text = strcat(text,"\n");
58      fprintf(fileID,text);
59  end
60
61  function myKeyPressFcn(hObject, event)
62      global KEY_IS_PRESSED
63      KEY_IS_PRESSED  = 1;
64      disp('key is pressed');
65  end
```

B.2.5 ExtractROIs_people.m

```
clear all
close all

MyFolderInfo = dir('frames_persona/GH010008_Val*');
i = 1;
rois =[];

for f = 401:543
    disp(f);
    rois_img = [];
    % Define new name for image
    string =
    ↪ strcat('./frames_persona/',MyFolderInfo(f).name);
    I = imread(string); imshow(I);

    % Select person
    roi = drawrectangle();

    % Crop and save person image
    I2 = imcrop(I,roi.Position);
    string =
    ↪ strcat('./frames_persona_cut/',MyFolderInfo(f).name);
    imwrite(I2,string);
    S = size(I2);
    pos = [2 2 S(2)-2 S(1)-2];

    % Save data
    rois_img = [rois_img ;pos];
    rois_p.name = string;
    rois_p.bboxes_pigs = [];
    rois_p.bboxes_pers = rois_img;
    rois = [rois rois_p];
    i = i+1;
end

% Save BBxs data
fileID = fopen('annotations_GH010008_Val_Pers_401to543.txt'
    ↪ ,'w');
for f = 1:size(rois,2)
```

```
37      text = rois(f).name;
38      for i = 1:size(rois(f).bboxes_pers,1)
39          text = strcat(text,"
    ↪   ",int2str(rois(f).bboxes_pers(i,1)),",",
    ↪   int2str(rois(f).bboxes_pers(i,2)),",", int2str(rois(f)
    ↪   .bboxes_pers(i,1)+rois(f).bboxes_pers(i,3)),",",
    ↪   int2str(rois(f).bboxes_pers(i,2)+rois(f).bboxes_pers(i
    ↪   ,4)),",1");
40      end
41      text = strcat(text,"\n");
42      fprintf(fileID,text);
43  end
44  fclose(fileID);
```

B.2.6 Object_Detection_DataPreprocessing.ipynb

```
import cv2
from matplotlib import pyplot as plt
import numpy as np
import os
import pandas as pd
import random
from skimage import io
from shutil import copyfile
import sys
import time

'''
Load data from .csv file
train-images-boxable.csv file contains the image name and image url
train-annotations-bbox.csv file contains the bounding box info with the image id (name) and the image label name
class-descriptions-boxable.csv file contains the image label name corresponding to its class name
Download link:

https://storage.googleapis.com/openimages/web/download.html'''

https://www.figure-eight.com/dataset/open-images-annotated-with-bounding-boxes/

!wget https://storage.googleapis.com/openimages/2018_04/train/train-annotations-bbox.csv
!wget https://storage.googleapis.com/openimages/2018_04/train/train-images-boxable-with-rotation.csv

'''
The original code used "train-images-boxable.csv" but I couldn't find it. So, I am using
"train-images-boxable-with-rotations.csv"
'''

```

```
!wget https://storage.googleapis.com/openimages/v5/class-descriptions-boxable.csv

images_boxable_fname = 'train-images-boxable-with-rotation.csv'
annotations_bbox_fname = 'train-annotations-bbox.csv'
class_descriptions_fname = 'class-descriptions-boxable.csv'

images_boxable = pd.read_csv(images_boxable_fname)
images_boxable.head()

annotations_bbox = pd.read_csv(annotations_bbox_fname)
annotations_bbox.head()

'''
1. **XMin, XMax, YMin, YMax**: coordinates of the box, in normalized image coordinates.
2. **IsOccluded**: Indicates that the object is occluded by another object in the image.
3. **IsTruncated**: Indicates that the object extends beyond the boundary of the image.
4. **IsGroupOf**: Indicates that the box spans a group of objects (e.g., a bed of flowers or a crowd of people). We asked annotators to use this tag for cases with more than 5 instances which are heavily occluding each other and are physically touching.
5. **IsDepiction**: Indicates that the object is a depiction (e.g., a cartoon or drawing of the object, not a real physical instance).
6. **IsInside**: Indicates a picture taken from the inside of the object (e.g., a car interior or inside of a building).
'''

class_descriptions = pd.read_csv(class_descriptions_fname, header=None)
class_descriptions.head()

### Plot Bounding box

def plot_bbox(img_id):
```

```
  img_url = images_boxable.loc[images_boxable["ImageID"]==img_id]['OriginalURL'].values[0]
  img = io.imread(img_url)
  height, width, channel = img.shape
  print(f"Image: {img.shape}")
  bboxs = annotations_bbox[annotations_bbox['ImageID']==img_id]
  for index, row in bboxs.iterrows():
      xmin = row['XMin']
      xmax = row['XMax']
      ymin = row['YMin']
      ymax = row['YMax']
      xmin = int(xmin*width)
      xmax = int(xmax*width)
      ymin = int(ymin*height)
      ymax = int(ymax*height)
      label_name = row['LabelName']
      class_series = class_descriptions[class_descriptions[0]==label_name]
      class_name = class_series[1].values[0]
      print(f"Coordinates: {xmin,ymin}, {xmax,ymax}")
      cv2.rectangle(img, (xmin,ymin), (xmax,ymax), (255,0,0), 5)
      font = cv2.FONT_HERSHEY_SIMPLEX
      cv2.putText(img, class_name, (xmin,ymin-10), font, 3, (0,255,0), 5)
  plt.figure(figsize=(15,10))
  plt.title('Image with Bounding Box')
  plt.imshow(img)
  plt.axis("off")
  plt.show()

'''
Finding images with lesser number of objects so as easy to visualize
'''

least_objects_img_ids = annotations_bbox["ImageID"].value_counts().tail(50).index.values

```

```
for img_id in random.sample(list(least_objects_img_ids), 5):
  plot_bbox(img_id)

### Get subset of the whole dataset

For here, I just want to detect three classes, which include person, mobile phone and car. We just extract 1000 images for each class from the whole dataset.

class_descriptions.loc[class_descriptions[1].isin(['Person'])]

# Find the label_name for 'Person', 'Mobile Phone' and 'Car' classes
person_pd = class_descriptions[class_descriptions[1]=='Person']

label_name_person = person_pd[0].values[0]

'''
**Be careful that there might be several object in one image. For example, there are three person and two mobile phone in one image**
'''

person_bbox = annotations_bbox[annotations_bbox['LabelName']==label_name_person]

print('There are %d persons in the dataset' %(len(person_bbox)))
person_img_id = person_bbox['ImageID']

person_bbox = annotations_bbox[annotations_bbox['LabelName']==label_name_person]

print('There are %d persons in the dataset' %(len(person_bbox)))
person_img_id = person_bbox['ImageID']

person_img_id = np.unique(person_img_id)
```

```
print('There are %d images which contain persons' % (len(person_img_id)))

'''
We just randomly pick n images in here.
'''

# here I've chosen only 10 images for speed, change it to your liking
n = 50
subperson_img_id = random.sample(list(person_img_id), n)
subperson_pd = images_boxable.loc[images_boxable['ImageID'].isin(subperson_img_id)]

subperson_pd.shape

subperson_pd.head()

subperson_dict = subperson_pd[["ImageID", "OriginalURL"]].set_index('ImageID')["OriginalURL"].to_dict()

mappings = [subperson_dict]

len(mappings)

len(mappings[0])

classes = ['Person']

### Download images

We need to save the images with filename as `image id` with jpg extension

# download images
num = 1
for idx, obj_type in enumerate(classes):
    n_issues = 0
    # create the directory
    if not os.path.exists(obj_type+"_mini"):
        os.mkdir(obj_type+"_mini")
```

```
155     for img_id, url in mappings[idx].items():
156         try:
157             if num%50 == 0:
158                 print(num)
159             num += 1
160             img = io.imread(url)
161             saved_path = os.path.join(obj_type+"_mini",
                ↪  img_id+".jpg")
162             io.imsave(saved_path, img)
163         except Exception as e:
164             n_issues += 1
165     print(f"Images Issues: {n_issues}")
166 
167 !ls Person_mini | wc -l
168 
169 ### Dataset format for Faster-RCNN code
170 
171 (fname_path, xmin, xmax, ymin, ymax, class_name)
172 
173 train: 0.8
174 validation: 0.2
175 
176 # save images to train and test directory
177 train_path = 'train_mini'
178 test_path = 'test_mini'
179 
180 !mkdir train_mini test_mini
181 
182 random.seed(1)
183 
184 for i in range(len(classes)):
185     all_imgs = os.listdir(classes[i]+"_mini")
186     all_imgs = [f for f in all_imgs if not
      ↪  f.startswith('.')]
187     random.shuffle(all_imgs)
188 
189     limit = int(n*0.8)
190 
191     train_imgs = all_imgs[:limit]
192     test_imgs = all_imgs[limit:]
193 
```

```
194     # copy each classes' images to train directory
195     for j in range(len(train_imgs)):
196         original_path = os.path.join(classes[i]+"_mini",
            ↪  train_imgs[j])
197         new_path = os.path.join(train_path, train_imgs[j])
198         copyfile(original_path, new_path)
199 
200     # copy each classes' images to test directory
201     for j in range(len(test_imgs)):
202         original_path = os.path.join(classes[i]+"_mini",
            ↪  test_imgs[j])
203         new_path = os.path.join(test_path, test_imgs[j])
204         copyfile(original_path, new_path)
205 
206 !ls train_mini | wc -l
207 
208 !ls test_mini | wc -l
209 
210 '''
211 The expected number of training images and validation
    ↪  images should be 24 and 6 respectively.
212 
213 However, there might be some overlap images which appear
    ↪  in two or three classes simultaneously. For instance,
    ↪  an image might be a person walking on the street and
    ↪  there are several cars in the street
214 '''
215 
216 label_names = [label_name_person]
217 
218 train_df = pd.DataFrame(columns=['FileName', 'XMin',
    ↪  'XMax', 'YMin', 'YMax', 'ClassName'])
219 
220 # Find boxes in each image and put them in a dataframe
221 train_imgs = os.listdir(train_path)
222 train_imgs = [name for name in train_imgs if not
    ↪  name.startswith('.')]
223 
224 for i in range(len(train_imgs)):
225     sys.stdout.write('Parse train_imgs ' + str(i) + ';
        ↪  Number of boxes: ' + str(len(train_df)) + '\r')
```

```
226     sys.stdout.flush()
227     img_name = train_imgs[i]
228     img_id = img_name[0:16]
229     tmp_df = annotations_bbox[annotations_bbox['ImageID']==
        ↪  img_id]
230     for index, row in tmp_df.iterrows():
231         labelName = row['LabelName']
232         for i in range(len(label_names)):
233             if labelName == label_names[i]:
234                 train_df = train_df.append({'FileName':
                    ↪  img_name,
235                                             'XMin': row['XM
                                                ↪  in'],
236                                             'XMax': row['XM
                                                ↪  ax'],
237                                             'YMin': row['YM
                                                ↪  in'],
238                                             'YMax': row['YM
                                                ↪  ax'],
239                                             'ClassName': cl
                                                ↪  asses[i]},
240                                            ignore_index=Tr
                                                ↪  ue)
241
242 train_df.head()
243
244 train_df.shape
245
246 '''
247 Let's test if they work fine by plotting the bounding box
    ↪  for the above 5 images
248 '''
249
250 train_img_ids = train_df["FileName"].head().str.split(".")
    ↪  .str[0].unique()
251
252 for img_id in train_img_ids:
253   plot_bbox(img_id)
254
255 '''
256 **This looks fine to me!!**
```

```
'''

test_df = pd.DataFrame(columns=['FileName', 'XMin', 'XMax', 'YMin', 'YMax', 'ClassName'])

# find boxes in each image and put them in a dataframe
test_imgs = os.listdir(test_path)
test_imgs = [name for name in test_imgs if not name.startswith('.')]

for i in range(len(test_imgs)):
    sys.stdout.write('Parse test_imgs ' + str(i) + '; Number of boxes: ' + str(len(test_df)) + '\r')
    sys.stdout.flush()
    img_name = test_imgs[i]
    img_id = img_name[0:16]
    tmp_df = annotations_bbox[annotations_bbox['ImageID']==img_id]
    for index, row in tmp_df.iterrows():
        labelName = row['LabelName']
        for i in range(len(label_names)):
            if labelName == label_names[i]:
                test_df = test_df.append({'FileName': img_name,
                                          'XMin': row['XMin'],
                                          'XMax': row['XMax'],
                                          'YMin': row['YMin'],
                                          'YMax': row['YMax'],
                                          'ClassName': classes[i]},
                                         ignore_index=True)

train_df.to_csv('train_mini.csv')
test_df.to_csv('test_mini.csv')

### Write train.csv to annotation.txt
```

```
287
288 train_df = pd.read_csv('train_mini.csv')
289
290 # for training
291 with open("annotation_mini.txt", "w+") as f:
292   for idx, row in train_df.iterrows():
293       sys.stdout.write(str(idx) + '\r')
294       sys.stdout.flush()
295       img = cv2.imread('train_mini/' + row['FileName'])
296       height, width = img.shape[:2]
297       x1 = int(row['XMin'] * width)
298       x2 = int(row['XMax'] * width)
299       y1 = int(row['YMin'] * height)
300       y2 = int(row['YMax'] * height)
301
302       google_colab_file_path = '/home/virginia/TFG/Faster
          ↪ R-CNN/Faster_RCNN_for_Open_Images_Dataset_Keras-
          ↪ master/train_mini'
303       fileName = os.path.join(google_colab_file_path,
          ↪ row['FileName'])
304       className = row['ClassName']
305       f.write(fileName + ',' + str(x1) + ',' + str(y1) +
          ↪ ',' + str(x2) + ',' + str(y2) + ',' + className +
          ↪ '\n')
306
307 test_df = pd.read_csv('test_mini.csv')
308
309 # for test
310 with open("test_annotation_mini.txt", "w+") as f:
311   for idx, row in test_df.iterrows():
312       sys.stdout.write(str(idx) + '\r')
313       sys.stdout.flush()
314       img = cv2.imread('test_mini/' + row['FileName'])
315       height, width = img.shape[:2]
316       x1 = int(row['XMin'] * width)
317       x2 = int(row['XMax'] * width)
318       y1 = int(row['YMin'] * height)
319       y2 = int(row['YMax'] * height)
320
```

```
321         google_colab_file_path = '/home/virginia/TFG/Faster
             ↪ R-CNN/Faster_RCNN_for_Open_Images_Dataset_Keras-
             ↪ master/test_mini'
322         fileName = os.path.join(google_colab_file_path,
             ↪ row['FileName'])
323         className = row['ClassName']
324         f.write(fileName + ',' + str(x1) + ',' + str(y1) +
             ↪ ',' + str(x2) + ',' + str(y2) + ',' + className +
             ↪ '\n')
```

C.3 Codi YOLOv4

C.3.1 configs.py

```
#================================================================
#
#   File name   : configs.py
#   Author      : PyLessons
#   Created date: 2020-08-18
#   Website     : https://pylessons.com/
#   GitHub      : https://github.com/pythonlessons/TensorFlow-2.x-YOLOv3
#   Description : yolov3 configuration file
#
#   Modified by : Virginia Ramón
#================================================================

# YOLO options
YOLO_TYPE                   = "yolov4" # yolov4 or yolov3
YOLO_FRAMEWORK              = "tf" # "tf" or "trt"
YOLO_V3_WEIGHTS             = "model_data/yolov3.weights"
YOLO_V4_WEIGHTS             = "model_data/yolov4.weights"
YOLO_V3_TINY_WEIGHTS        = "model_data/yolov3-tiny.weights"
YOLO_V4_TINY_WEIGHTS        = "model_data/yolov4-tiny.weights"
YOLO_TRT_QUANTIZE_MODE      = "INT8" # INT8, FP16, FP32
#YOLO_CUSTOM_WEIGHTS         = False
#YOLO_COCO_CLASSES           = "model_data/coco/coco.names"
YOLO_CUSTOM_WEIGHTS         = True
YOLO_COCO_CLASSES           = "/home/virginia/TFG/YOLOv4/TensorFlow-2.x-YOLOv3-master/data/annotation.names"
YOLO_STRIDES                = [8, 16, 32]
YOLO_IOU_LOSS_THRESH        = 0.9
YOLO_ANCHOR_PER_SCALE       = 3
YOLO_MAX_BBOX_PER_SCALE     = 100
YOLO_INPUT_SIZE             = 416
if YOLO_TYPE                == "yolov4":
```

```
    YOLO_ANCHORS            = [[[12,  16], [19,   36],
     ↪  [40,   28]],
                                [[36,  75], [76,   55],
                                 ↪  [72,  146]],
                                [[142,110], [192, 243],
                                 ↪  [459, 401]]]
if YOLO_TYPE                == "yolov3":
    YOLO_ANCHORS            = [[[10,  13], [16,   30],
     ↪  [33,   23]],
                                [[30,  61], [62,   45],
                                 ↪  [59,  119]],
                                [[116, 90], [156, 198],
                                 ↪  [373, 326]]]
# Train options
TRAIN_YOLO_TINY             = False
TRAIN_SAVE_BEST_ONLY        = True # saves only best model
 ↪  according validation loss (True recommended)
TRAIN_SAVE_CHECKPOINT       = False # saves all best
 ↪  validated checkpoints in training process (may require
 ↪  a lot disk space) (False recommended)
TRAIN_CLASSES               = "/home/virginia/TFG/YOLOv4/T
 ↪  ensorFlow-2.x-YOLOv3-master/data/annotation.names"
TRAIN_ANNOT_PATH            =
 ↪  "/home/virginia/TFG/YOLOv4/TensorFlow-2.x-YOLOv3-maste
 ↪  r/data/PigPersv2/annotation_train.txt"
TRAIN_LOGDIR                = "log"
TRAIN_CHECKPOINTS_FOLDER    =
 ↪  "checkpoints/yolov4_PigPerse46"
TRAIN_MODEL_NAME            = f"{YOLO_TYPE}_custom"
TRAIN_LOAD_IMAGES_TO_RAM    = True # With True faster
 ↪  training, but need more RAM
TRAIN_BATCH_SIZE            = 4
TRAIN_INPUT_SIZE            = 416
TRAIN_DATA_AUG              = True
TRAIN_TRANSFER              = True
TRAIN_FROM_CHECKPOINT       = False #
 ↪  "checkpoints/yolov3_custom"
TRAIN_LR_INIT               = 4e-6
TRAIN_LR_END                = 4e-7
TRAIN_WARMUP_EPOCHS         = 1
TRAIN_EPOCHS                = 40
```

```
57
58 # TEST options
59 TEST_ANNOT_PATH               =
   ↪  "/home/virginia/TFG/YOLOv4/TensorFlow-2.x-YOLOv3-maste┘
   ↪  r/data/PigPersv2/annotation_val.txt"
60 TEST_BATCH_SIZE              = 4
61 TEST_INPUT_SIZE              = 416
62 TEST_DATA_AUG                = False
63 TEST_DECTECTED_IMAGE_PATH    = ""
64 TEST_SCORE_THRESHOLD         = 0.5
65 TEST_IOU_THRESHOLD           = 0.5
66
67 if TRAIN_YOLO_TINY:
68     YOLO_STRIDES              = [16, 32]
69     # YOLO_ANCHORS              = [[[23, 27],  [37, 58],
       ↪  [81,  82]], # this line can be uncommented for
       ↪  default coco weights
70     YOLO_ANCHORS              = [[[10, 14],  [23, 27],
       ↪  [37, 58]],
71                                  [[81,  82], [135, 169],
                                    ↪  [344, 319]]]
```

C.3.2 yolov4.py

```
#================================================================
#
#   File name   : yolov4.py
#   Author      : PyLessons
#   Created date: 2020-09-31
#   Website     : https://pylessons.com/
#   GitHub      : https://github.com/pythonlessons/TensorFlow-2.x-YOLOv3
#   Description : main yolov3 & yolov4 functions
#
#================================================================
import numpy as np
import tensorflow as tf
from tensorflow.keras.layers import Conv2D, Input, LeakyReLU, ZeroPadding2D, BatchNormalization, MaxPool2D
from tensorflow.keras.regularizers import l2
from yolov3.configs import *
import sys
sys.path.append("/home/virginia/TFG/YOLOv4/TensorFlow-2.x-YOLOv3-master/")

STRIDES         = np.array(YOLO_STRIDES)
ANCHORS         = (np.array(YOLO_ANCHORS).T/STRIDES).T

def read_class_names(class_file_name):
    # loads class name from a file
    names = {}
    with open(class_file_name, 'r') as data:
        for ID, name in enumerate(data):
            names[ID] = name.strip('\n')
    return names

class BatchNormalization(BatchNormalization):
    # "Frozen state" and "inference mode" are two separate concepts.
    # `layer.trainable = False` is to freeze the layer, so the layer will use
```

```
33      # stored moving `var` and `mean` in the "inference
        ↪ mode", and both `gama`
34      # and `beta` will not be updated !
35      def call(self, x, training=False):
36          if not training:
37              training = tf.constant(False)
38          training = tf.logical_and(training, self.trainable)
39          return super().call(x, training)
40
41  def convolutional(input_layer, filters_shape,
    ↪ downsample=False, activate=True, bn=True,
    ↪ activate_type='leaky'):
42      if downsample:
43          input_layer = ZeroPadding2D(((1, 0), (1,
            ↪ 0)))(input_layer)
44          padding = 'valid'
45          strides = 2
46      else:
47          strides = 1
48          padding = 'same'
49
50      conv = Conv2D(filters=filters_shape[-1], kernel_size =
        ↪ filters_shape[0], strides=strides,
51                    padding=padding, use_bias=not bn,
                      ↪ kernel_regularizer=l2(0.0005),
52                    kernel_initializer=tf.random_normal_init
                      ↪ ializer(stddev=0.01),
53                    bias_initializer=tf.constant_initializer
                      ↪ (0.))(input_layer)
54      if bn:
55          conv = BatchNormalization()(conv)
56      if activate == True:
57          if activate_type == "leaky":
58              conv = LeakyReLU(alpha=0.1)(conv)
59          elif activate_type == "mish":
60              conv = mish(conv)
61
62      return conv
63
64  def mish(x):
65      return x * tf.math.tanh(tf.math.softplus(x))
```

```
66
67 def residual_block(input_layer, input_channel,
   ↪  filter_num1, filter_num2, activate_type='leaky'):
68     short_cut = input_layer
69     conv = convolutional(input_layer, filters_shape=(1, 1,
       ↪  input_channel, filter_num1),
       ↪  activate_type=activate_type)
70     conv = convolutional(conv       , filters_shape=(3, 3,
       ↪  filter_num1,   filter_num2),
       ↪  activate_type=activate_type)
71
72     residual_output = short_cut + conv
73     return residual_output
74
75 def upsample(input_layer):
76     return tf.image.resize(input_layer,
       ↪  (input_layer.shape[1] * 2, input_layer.shape[2] *
       ↪  2), method='nearest')
77
78 def route_group(input_layer, groups, group_id):
79     convs = tf.split(input_layer,
       ↪  num_or_size_splits=groups, axis=-1)
80     return convs[group_id]
81
82 def darknet53(input_data):
83     input_data = convolutional(input_data, (3, 3,  3,  32))
84     input_data = convolutional(input_data, (3, 3, 32,
       ↪  64), downsample=True)
85
86     for i in range(1):
87         input_data = residual_block(input_data,  64,  32,
           ↪  64)
88
89     input_data = convolutional(input_data, (3, 3,  64,
       ↪  128), downsample=True)
90
91     for i in range(2):
92         input_data = residual_block(input_data, 128,  64,
           ↪  128)
93
```

```
94      input_data = convolutional(input_data, (3, 3, 128,
        ↪  256), downsample=True)
95
96      for i in range(8):
97          input_data = residual_block(input_data, 256, 128,
            ↪  256)
98
99      route_1 = input_data
100     input_data = convolutional(input_data, (3, 3, 256,
        ↪  512), downsample=True)
101
102     for i in range(8):
103         input_data = residual_block(input_data, 512, 256,
            ↪  512)
104
105     route_2 = input_data
106     input_data = convolutional(input_data, (3, 3, 512,
        ↪  1024), downsample=True)
107
108     for i in range(4):
109         input_data = residual_block(input_data, 1024, 512,
            ↪  1024)
110
111     return route_1, route_2, input_data
112
113 def cspdarknet53(input_data):
114     input_data = convolutional(input_data, (3, 3,  3,
        ↪  32), activate_type="mish")
115     input_data = convolutional(input_data, (3, 3, 32,
        ↪  64), downsample=True, activate_type="mish")
116
117     route = input_data
118     route = convolutional(route, (1, 1, 64, 64),
        ↪  activate_type="mish")
119     input_data = convolutional(input_data, (1, 1, 64, 64),
        ↪  activate_type="mish")
120     for i in range(1):
121         input_data = residual_block(input_data,  64,  32,
            ↪  64, activate_type="mish")
122     input_data = convolutional(input_data, (1, 1, 64, 64),
        ↪  activate_type="mish")
```

```
123
124     input_data = tf.concat([input_data, route], axis=-1)
125     input_data = convolutional(input_data, (1, 1, 128,
        ↪  64), activate_type="mish")
126     input_data = convolutional(input_data, (3, 3, 64,
        ↪  128), downsample=True, activate_type="mish")
127     route = input_data
128     route = convolutional(route, (1, 1, 128, 64),
        ↪  activate_type="mish")
129     input_data = convolutional(input_data, (1, 1, 128,
        ↪  64), activate_type="mish")
130     for i in range(2):
131         input_data = residual_block(input_data, 64,  64,
            ↪  64, activate_type="mish")
132     input_data = convolutional(input_data, (1, 1, 64, 64),
        ↪  activate_type="mish")
133     input_data = tf.concat([input_data, route], axis=-1)
134
135     input_data = convolutional(input_data, (1, 1, 128,
        ↪  128), activate_type="mish")
136     input_data = convolutional(input_data, (3, 3, 128,
        ↪  256), downsample=True, activate_type="mish")
137     route = input_data
138     route = convolutional(route, (1, 1, 256, 128),
        ↪  activate_type="mish")
139     input_data = convolutional(input_data, (1, 1, 256,
        ↪  128), activate_type="mish")
140     for i in range(8):
141         input_data = residual_block(input_data, 128, 128,
            ↪  128, activate_type="mish")
142     input_data = convolutional(input_data, (1, 1, 128,
        ↪  128), activate_type="mish")
143     input_data = tf.concat([input_data, route], axis=-1)
144
145     input_data = convolutional(input_data, (1, 1, 256,
        ↪  256), activate_type="mish")
146     route_1 = input_data
147     input_data = convolutional(input_data, (3, 3, 256,
        ↪  512), downsample=True, activate_type="mish")
148     route = input_data
```

```
149     route = convolutional(route, (1, 1, 512, 256),
        ↪  activate_type="mish")
150     input_data = convolutional(input_data, (1, 1, 512,
        ↪  256), activate_type="mish")
151     for i in range(8):
152         input_data = residual_block(input_data, 256, 256,
            ↪  256, activate_type="mish")
153     input_data = convolutional(input_data, (1, 1, 256,
        ↪  256), activate_type="mish")
154     input_data = tf.concat([input_data, route], axis=-1)
155 
156     input_data = convolutional(input_data, (1, 1, 512,
        ↪  512), activate_type="mish")
157     route_2 = input_data
158     input_data = convolutional(input_data, (3, 3, 512,
        ↪  1024), downsample=True, activate_type="mish")
159     route = input_data
160     route = convolutional(route, (1, 1, 1024, 512),
        ↪  activate_type="mish")
161     input_data = convolutional(input_data, (1, 1, 1024,
        ↪  512), activate_type="mish")
162     for i in range(4):
163         input_data = residual_block(input_data, 512, 512,
            ↪  512, activate_type="mish")
164     input_data = convolutional(input_data, (1, 1, 512,
        ↪  512), activate_type="mish")
165     input_data = tf.concat([input_data, route], axis=-1)
166 
167     input_data = convolutional(input_data, (1, 1, 1024,
        ↪  1024), activate_type="mish")
168     input_data = convolutional(input_data, (1, 1, 1024,
        ↪  512))
169     input_data = convolutional(input_data, (3, 3, 512,
        ↪  1024))
170     input_data = convolutional(input_data, (1, 1, 1024,
        ↪  512))
171 
172     max_pooling_1 =
        ↪  tf.keras.layers.MaxPool2D(pool_size=13,
        ↪  padding='SAME', strides=1)(input_data)
```

```
173     max_pooling_2 = tf.keras.layers.MaxPool2D(pool_size=9,
        ↪ padding='SAME', strides=1)(input_data)
174     max_pooling_3 = tf.keras.layers.MaxPool2D(pool_size=5,
        ↪ padding='SAME', strides=1)(input_data)
175     input_data = tf.concat([max_pooling_1, max_pooling_2,
        ↪ max_pooling_3, input_data], axis=-1)
176
177     input_data = convolutional(input_data, (1, 1, 2048,
        ↪ 512))
178     input_data = convolutional(input_data, (3, 3, 512,
        ↪ 1024))
179     input_data = convolutional(input_data, (1, 1, 1024,
        ↪ 512))
180
181     return route_1, route_2, input_data
182
183 def darknet19_tiny(input_data):
184     input_data = convolutional(input_data, (3, 3, 3, 16))
185     input_data = MaxPool2D(2, 2, 'same')(input_data)
186     input_data = convolutional(input_data, (3, 3, 16, 32))
187     input_data = MaxPool2D(2, 2, 'same')(input_data)
188     input_data = convolutional(input_data, (3, 3, 32, 64))
189     input_data = MaxPool2D(2, 2, 'same')(input_data)
190     input_data = convolutional(input_data, (3, 3, 64, 128))
191     input_data = MaxPool2D(2, 2, 'same')(input_data)
192     input_data = convolutional(input_data, (3, 3, 128,
        ↪ 256))
193     route_1 = input_data
194     input_data = MaxPool2D(2, 2, 'same')(input_data)
195     input_data = convolutional(input_data, (3, 3, 256,
        ↪ 512))
196     input_data = MaxPool2D(2, 1, 'same')(input_data)
197     input_data = convolutional(input_data, (3, 3, 512,
        ↪ 1024))
198
199     return route_1, input_data
200
201 def cspdarknet53_tiny(input_data): # not sure how this
    ↪ should be called
202     input_data = convolutional(input_data, (3, 3, 3, 32),
        ↪ downsample=True)
```

```
203     input_data = convolutional(input_data, (3, 3, 32, 64),
        ↪  downsample=True)
204     input_data = convolutional(input_data, (3, 3, 64, 64))
205
206     route = input_data
207     input_data = route_group(input_data, 2, 1)
208     input_data = convolutional(input_data, (3, 3, 32, 32))
209     route_1 = input_data
210     input_data = convolutional(input_data, (3, 3, 32, 32))
211     input_data = tf.concat([input_data, route_1], axis=-1)
212     input_data = convolutional(input_data, (1, 1, 32, 64))
213     input_data = tf.concat([route, input_data], axis=-1)
214     input_data = MaxPool2D(2, 2, 'same')(input_data)
215
216     input_data = convolutional(input_data, (3, 3, 64, 128))
217     route = input_data
218     input_data = route_group(input_data, 2, 1)
219     input_data = convolutional(input_data, (3, 3, 64, 64))
220     route_1 = input_data
221     input_data = convolutional(input_data, (3, 3, 64, 64))
222     input_data = tf.concat([input_data, route_1], axis=-1)
223     input_data = convolutional(input_data, (1, 1, 64, 128))
224     input_data = tf.concat([route, input_data], axis=-1)
225     input_data = MaxPool2D(2, 2, 'same')(input_data)
226
227     input_data = convolutional(input_data, (3, 3, 128,
        ↪  256))
228     route = input_data
229     input_data = route_group(input_data, 2, 1)
230     input_data = convolutional(input_data, (3, 3, 128,
        ↪  128))
231     route_1 = input_data
232     input_data = convolutional(input_data, (3, 3, 128,
        ↪  128))
233     input_data = tf.concat([input_data, route_1], axis=-1)
234     input_data = convolutional(input_data, (1, 1, 128,
        ↪  256))
235     route_1 = input_data
236     input_data = tf.concat([route, input_data], axis=-1)
237     input_data = MaxPool2D(2, 2, 'same')(input_data)
238
```

```
239     input_data = convolutional(input_data, (3, 3, 512,
        ↪  512))
240
241     return route_1, input_data
242
243 def YOLOv3(input_layer, NUM_CLASS):
244     # After the input layer enters the Darknet-53 network,
        ↪  we get three branches
245     route_1, route_2, conv = darknet53(input_layer)
246     # See the orange module (DBL) in the figure above, a
        ↪  total of 5 Subconvolution operation
247     conv = convolutional(conv, (1, 1, 1024,  512))
248     conv = convolutional(conv, (3, 3,  512, 1024))
249     conv = convolutional(conv, (1, 1, 1024,  512))
250     conv = convolutional(conv, (3, 3,  512, 1024))
251     conv = convolutional(conv, (1, 1, 1024,  512))
252     conv_lobj_branch = convolutional(conv, (3, 3, 512,
        ↪  1024))
253
254     # conv_lbbox is used to predict large-sized objects ,
        ↪  Shape = [None, 13, 13, 255]
255     conv_lbbox = convolutional(conv_lobj_branch, (1, 1,
        ↪  1024, 3*(NUM_CLASS + 5)), activate=False, bn=False)
256
257     conv = convolutional(conv, (1, 1,  512,  256))
258     # upsample here uses the nearest neighbor
        ↪  interpolation method, which has the advantage that
        ↪  the
259     # upsampling process does not need to learn, thereby
        ↪  reducing the network parameter
260     conv = upsample(conv)
261
262     conv = tf.concat([conv, route_2], axis=-1)
263     conv = convolutional(conv, (1, 1, 768, 256))
264     conv = convolutional(conv, (3, 3, 256, 512))
265     conv = convolutional(conv, (1, 1, 512, 256))
266     conv = convolutional(conv, (3, 3, 256, 512))
267     conv = convolutional(conv, (1, 1, 512, 256))
268     conv_mobj_branch = convolutional(conv, (3, 3, 256,
        ↪  512))
269
```

```
270     # conv_mbbox is used to predict medium-sized objects,
        ↪ shape = [None, 26, 26, 255]
271     conv_mbbox = convolutional(conv_mobj_branch, (1, 1,
        ↪ 512, 3*(NUM_CLASS + 5)), activate=False, bn=False)
272
273     conv = convolutional(conv, (1, 1, 256, 128))
274     conv = upsample(conv)
275
276     conv = tf.concat([conv, route_1], axis=-1)
277     conv = convolutional(conv, (1, 1, 384, 128))
278     conv = convolutional(conv, (3, 3, 128, 256))
279     conv = convolutional(conv, (1, 1, 256, 128))
280     conv = convolutional(conv, (3, 3, 128, 256))
281     conv = convolutional(conv, (1, 1, 256, 128))
282     conv_sobj_branch = convolutional(conv, (3, 3, 128,
        ↪ 256))
283
284     # conv_sbbox is used to predict small size objects,
        ↪ shape = [None, 52, 52, 255]
285     conv_sbbox = convolutional(conv_sobj_branch, (1, 1,
        ↪ 256, 3*(NUM_CLASS +5)), activate=False, bn=False)
286
287     return [conv_sbbox, conv_mbbox, conv_lbbox]
288
289 def YOLOv4(input_layer, NUM_CLASS):
290     route_1, route_2, conv = cspdarknet53(input_layer)
291
292     route = conv
293     conv = convolutional(conv, (1, 1, 512, 256))
294     conv = upsample(conv)
295     route_2 = convolutional(route_2, (1, 1, 512, 256))
296     conv = tf.concat([route_2, conv], axis=-1)
297
298     conv = convolutional(conv, (1, 1, 512, 256))
299     conv = convolutional(conv, (3, 3, 256, 512))
300     conv = convolutional(conv, (1, 1, 512, 256))
301     conv = convolutional(conv, (3, 3, 256, 512))
302     conv = convolutional(conv, (1, 1, 512, 256))
303
304     route_2 = conv
305     conv = convolutional(conv, (1, 1, 256, 128))
```

```
306     conv = upsample(conv)
307     route_1 = convolutional(route_1, (1, 1, 256, 128))
308     conv = tf.concat([route_1, conv], axis=-1)
309 
310     conv = convolutional(conv, (1, 1, 256, 128))
311     conv = convolutional(conv, (3, 3, 128, 256))
312     conv = convolutional(conv, (1, 1, 256, 128))
313     conv = convolutional(conv, (3, 3, 128, 256))
314     conv = convolutional(conv, (1, 1, 256, 128))
315 
316     route_1 = conv
317     conv = convolutional(conv, (3, 3, 128, 256))
318     conv_sbbox = convolutional(conv, (1, 1, 256, 3 *
        ↪  (NUM_CLASS + 5)), activate=False, bn=False)
319 
320     conv = convolutional(route_1, (3, 3, 128, 256),
        ↪  downsample=True)
321     conv = tf.concat([conv, route_2], axis=-1)
322 
323     conv = convolutional(conv, (1, 1, 512, 256))
324     conv = convolutional(conv, (3, 3, 256, 512))
325     conv = convolutional(conv, (1, 1, 512, 256))
326     conv = convolutional(conv, (3, 3, 256, 512))
327     conv = convolutional(conv, (1, 1, 512, 256))
328 
329     route_2 = conv
330     conv = convolutional(conv, (3, 3, 256, 512))
331     conv_mbbox = convolutional(conv, (1, 1, 512, 3 *
        ↪  (NUM_CLASS + 5)), activate=False, bn=False)
332 
333     conv = convolutional(route_2, (3, 3, 256, 512),
        ↪  downsample=True)
334     conv = tf.concat([conv, route], axis=-1)
335 
336     conv = convolutional(conv, (1, 1, 1024, 512))
337     conv = convolutional(conv, (3, 3, 512, 1024))
338     conv = convolutional(conv, (1, 1, 1024, 512))
339     conv = convolutional(conv, (3, 3, 512, 1024))
340     conv = convolutional(conv, (1, 1, 1024, 512))
341 
342     conv = convolutional(conv, (3, 3, 512, 1024))
```

```
343     conv_lbbox = convolutional(conv, (1, 1, 1024, 3 *
        ↪  (NUM_CLASS + 5)), activate=False, bn=False)
344
345     return [conv_sbbox, conv_mbbox, conv_lbbox]
346
347 def YOLOv3_tiny(input_layer, NUM_CLASS):
348     # After the input layer enters the Darknet-53 network,
        ↪  we get three branches
349     route_1, conv = darknet19_tiny(input_layer)
350
351     conv = convolutional(conv, (1, 1, 1024, 256))
352     conv_lobj_branch = convolutional(conv, (3, 3, 256,
        ↪  512))
353
354     # conv_lbbox is used to predict large-sized objects ,
        ↪  Shape = [None, 26, 26, 255]
355     conv_lbbox = convolutional(conv_lobj_branch, (1, 1,
        ↪  512, 3*(NUM_CLASS + 5)), activate=False, bn=False)
356
357     conv = convolutional(conv, (1, 1, 256, 128))
358     # upsample here uses the nearest neighbor
        ↪  interpolation method, which has the advantage that
        ↪  the
359     # upsampling process does not need to learn, thereby
        ↪  reducing the network parameter
360     conv = upsample(conv)
361
362     conv = tf.concat([conv, route_1], axis=-1)
363     conv_mobj_branch = convolutional(conv, (3, 3, 128,
        ↪  256))
364     # conv_mbbox is used to predict medium size objects,
        ↪  shape = [None, 13, 13, 255]
365     conv_mbbox = convolutional(conv_mobj_branch, (1, 1,
        ↪  256, 3 * (NUM_CLASS + 5)), activate=False,
        ↪  bn=False)
366
367     return [conv_mbbox, conv_lbbox]
368
369 def YOLOv4_tiny(input_layer, NUM_CLASS):
370     route_1, conv = cspdarknet53_tiny(input_layer)
371
```

```
372     conv = convolutional(conv, (1, 1, 512, 256))
373
374     conv_lobj_branch = convolutional(conv, (3, 3, 256,
        ↪ 512))
375     conv_lbbox = convolutional(conv_lobj_branch, (1, 1,
        ↪ 512, 3 * (NUM_CLASS + 5)), activate=False,
        ↪ bn=False)
376
377     conv = convolutional(conv, (1, 1, 256, 128))
378     conv = upsample(conv)
379     conv = tf.concat([conv, route_1], axis=-1)
380
381     conv_mobj_branch = convolutional(conv, (3, 3, 128,
        ↪ 256))
382     conv_mbbox = convolutional(conv_mobj_branch, (1, 1,
        ↪ 256, 3 * (NUM_CLASS + 5)), activate=False,
        ↪ bn=False)
383
384     return [conv_mbbox, conv_lbbox]
385
386 def Create_Yolo(input_size=416, channels=3,
    ↪ training=False, CLASSES=YOLO_COCO_CLASSES):
387     NUM_CLASS = len(read_class_names(CLASSES))
388     input_layer  = Input([input_size, input_size,
        ↪ channels])
389
390     if TRAIN_YOLO_TINY:
391         if YOLO_TYPE == "yolov4":
392             conv_tensors = YOLOv4_tiny(input_layer,
                ↪ NUM_CLASS)
393         if YOLO_TYPE == "yolov3":
394             conv_tensors = YOLOv3_tiny(input_layer,
                ↪ NUM_CLASS)
395     else:
396         if YOLO_TYPE == "yolov4":
397             conv_tensors = YOLOv4(input_layer, NUM_CLASS)
398         if YOLO_TYPE == "yolov3":
399             conv_tensors = YOLOv3(input_layer, NUM_CLASS)
400
401     output_tensors = []
402     for i, conv_tensor in enumerate(conv_tensors):
```

```
403         pred_tensor = decode(conv_tensor, NUM_CLASS, i)
404         if training: output_tensors.append(conv_tensor)
405         output_tensors.append(pred_tensor)
406 
407     Yolo = tf.keras.Model(input_layer, output_tensors)
408     return Yolo
409 
410 
411 def decode(conv_output, NUM_CLASS, i=0):
412     # where i = 0, 1 or 2 to correspond to the three grid
        ↪ scales
413     conv_shape       = tf.shape(conv_output)
414     batch_size       = conv_shape[0]
415     output_size      = conv_shape[1]
416 
417     conv_output = tf.reshape(conv_output, (batch_size,
        ↪ output_size, output_size, 3, 5 + NUM_CLASS))
418 
419     #conv_raw_dxdy = conv_output[:, :, :, :, 0:2] # offset
        ↪ of center position
420     #conv_raw_dwdh = conv_output[:, :, :, :, 2:4] #
        ↪ Prediction box length and width offset
421     #conv_raw_conf = conv_output[:, :, :, :, 4:5] #
        ↪ confidence of the prediction box
422     #conv_raw_prob = conv_output[:, :, :, :, 5: ] #
        ↪ category probability of the prediction box
423     conv_raw_dxdy, conv_raw_dwdh, conv_raw_conf,
        ↪ conv_raw_prob = tf.split(conv_output, (2, 2, 1,
        ↪ NUM_CLASS), axis=-1)
424 
425     # next need Draw the grid. Where output_size is equal
        ↪ to 13, 26 or 52
426     #y = tf.range(output_size, dtype=tf.int32)
427     #y = tf.expand_dims(y, -1)
428     #y = tf.tile(y, [1, output_size])
429     #x = tf.range(output_size,dtype=tf.int32)
430     #x = tf.expand_dims(x, 0)
431     #x = tf.tile(x, [output_size, 1])
432     xy_grid = tf.meshgrid(tf.range(output_size),
        ↪ tf.range(output_size))
```

```
433     xy_grid = tf.expand_dims(tf.stack(xy_grid, axis=-1),
        ↪  axis=2)  # [gx, gy, 1, 2]
434     xy_grid = tf.tile(tf.expand_dims(xy_grid, axis=0),
        ↪  [batch_size, 1, 1, 3, 1])
435     xy_grid = tf.cast(xy_grid, tf.float32)
436 
437     #xy_grid = tf.concat([x[:, :, tf.newaxis], y[:, :,
        ↪  tf.newaxis]], axis=-1)
438     #xy_grid = tf.tile(xy_grid[tf.newaxis, :, :,
        ↪  tf.newaxis, :], [batch_size, 1, 1, 3, 1])
439     #y_grid = tf.cast(xy_grid, tf.float32)
440 
441     # Calculate the center position of the prediction box:
442     pred_xy = (tf.sigmoid(conv_raw_dxdy) + xy_grid) *
        ↪  STRIDES[i]
443     # Calculate the length and width of the prediction box:
444     pred_wh = (tf.exp(conv_raw_dwdh) * ANCHORS[i]) *
        ↪  STRIDES[i]
445 
446     pred_xywh = tf.concat([pred_xy, pred_wh], axis=-1)
447     pred_conf = tf.sigmoid(conv_raw_conf) # object box
        ↪  calculates the predicted confidence
448     pred_prob = tf.sigmoid(conv_raw_prob) # calculating
        ↪  the predicted probability category box object
449 
450     # calculating the predicted probability category box
        ↪  object
451     return tf.concat([pred_xywh, pred_conf, pred_prob],
        ↪  axis=-1)
452 
453 
454 def bbox_iou(boxes1, boxes2):
455     boxes1_area = boxes1[..., 2] * boxes1[..., 3]
456     boxes2_area = boxes2[..., 2] * boxes2[..., 3]
457 
458     boxes1 = tf.concat([boxes1[..., :2] - boxes1[..., 2:]
        ↪  * 0.5,
459                         boxes1[..., :2] + boxes1[..., 2:]
                            ↪  * 0.5], axis=-1)
460     boxes2 = tf.concat([boxes2[..., :2] - boxes2[..., 2:]
        ↪  * 0.5,
```

```
461                         boxes2[..., :2] + boxes2[..., 2:]
                             ↪  * 0.5], axis=-1)
462
463     left_up = tf.maximum(boxes1[..., :2], boxes2[..., :2])
464     right_down = tf.minimum(boxes1[..., 2:], boxes2[...,
         ↪  2:])
465
466     inter_section = tf.maximum(right_down - left_up, 0.0)
467     inter_area = inter_section[..., 0] *
         ↪  inter_section[..., 1]
468     union_area = boxes1_area + boxes2_area - inter_area
469
470     return 1.0 * inter_area / union_area
471
472 def bbox_giou(boxes1, boxes2):
473     boxes1 = tf.concat([boxes1[..., :2] - boxes1[..., 2:]
         ↪  * 0.5,
474                         boxes1[..., :2] + boxes1[..., 2:]
                             ↪  * 0.5], axis=-1)
475     boxes2 = tf.concat([boxes2[..., :2] - boxes2[..., 2:]
         ↪  * 0.5,
476                         boxes2[..., :2] + boxes2[..., 2:]
                             ↪  * 0.5], axis=-1)
477
478     boxes1 = tf.concat([tf.minimum(boxes1[..., :2],
         ↪  boxes1[..., 2:]),
479                         tf.maximum(boxes1[..., :2],
                             ↪  boxes1[..., 2:])], axis=-1)
480     boxes2 = tf.concat([tf.minimum(boxes2[..., :2],
         ↪  boxes2[..., 2:]),
481                         tf.maximum(boxes2[..., :2],
                             ↪  boxes2[..., 2:])], axis=-1)
482
483     boxes1_area = (boxes1[..., 2] - boxes1[..., 0]) *
         ↪  (boxes1[..., 3] - boxes1[..., 1])
484     boxes2_area = (boxes2[..., 2] - boxes2[..., 0]) *
         ↪  (boxes2[..., 3] - boxes2[..., 1])
485
486     left_up = tf.maximum(boxes1[..., :2], boxes2[..., :2])
487     right_down = tf.minimum(boxes1[..., 2:], boxes2[...,
         ↪  2:])
```

```
488
489     inter_section = tf.maximum(right_down - left_up, 0.0)
490     inter_area = inter_section[..., 0] *
        ↪ inter_section[..., 1]
491     union_area = boxes1_area + boxes2_area - inter_area
492
493     # Calculate the iou value between the two bounding
        ↪ boxes
494     iou = inter_area / union_area
495
496     # Calculate the coordinates of the upper left corner
        ↪ and the lower right corner of the smallest closed
        ↪ convex surface
497     enclose_left_up = tf.minimum(boxes1[..., :2],
        ↪ boxes2[..., :2])
498     enclose_right_down = tf.maximum(boxes1[..., 2:],
        ↪ boxes2[..., 2:])
499     enclose = tf.maximum(enclose_right_down -
        ↪ enclose_left_up, 0.0)
500
501     # Calculate the area of the smallest closed convex
        ↪ surface C
502     enclose_area = enclose[..., 0] * enclose[..., 1]
503
504     # Calculate the GIoU value according to the GioU
        ↪ formula
505     giou = iou - 1.0 * (enclose_area - union_area) /
        ↪ enclose_area
506
507     return giou
508
509 # testing (should be better than giou)
510 def bbox_ciou(boxes1, boxes2):
511     boxes1_coor = tf.concat([boxes1[..., :2] - boxes1[...,
        ↪ 2:] * 0.5,
512                              boxes1[..., :2] + boxes1[..., 2:]
                                 ↪ * 0.5], axis=-1)
513     boxes2_coor = tf.concat([boxes2[..., :2] - boxes2[...,
        ↪ 2:] * 0.5,
514                              boxes2[..., :2] + boxes2[..., 2:]
                                 ↪ * 0.5], axis=-1)
```

```
515
516     left = tf.maximum(boxes1_coor[..., 0],
        ↪  boxes2_coor[..., 0])
517     up = tf.maximum(boxes1_coor[..., 1], boxes2_coor[...,
        ↪  1])
518     right = tf.maximum(boxes1_coor[..., 2],
        ↪  boxes2_coor[..., 2])
519     down = tf.maximum(boxes1_coor[..., 3],
        ↪  boxes2_coor[..., 3])
520
521     c = (right - left) * (right - left) + (up - down) *
        ↪  (up - down)
522     iou = bbox_iou(boxes1, boxes2)
523
524     u = (boxes1[..., 0] - boxes2[..., 0]) * (boxes1[...,
        ↪  0] - boxes2[..., 0]) + (boxes1[..., 1] -
        ↪  boxes2[..., 1]) * (boxes1[..., 1] - boxes2[..., 1])
525     d = u / c
526
527     ar_gt = boxes2[..., 2] / boxes2[..., 3]
528     ar_pred = boxes1[..., 2] / boxes1[..., 3]
529
530     ar_loss = 4 / (np.pi * np.pi) * (tf.atan(ar_gt) -
        ↪  tf.atan(ar_pred)) * (tf.atan(ar_gt) -
        ↪  tf.atan(ar_pred))
531     alpha = ar_loss / (1 - iou + ar_loss + 0.000001)
532     ciou_term = d + alpha * ar_loss
533
534     return iou - ciou_term
535
536
537 def compute_loss(pred, conv, label, bboxes, i=0,
    ↪  CLASSES=YOLO_COCO_CLASSES):
538     NUM_CLASS = len(read_class_names(CLASSES))
539     conv_shape  = tf.shape(conv)
540     batch_size  = conv_shape[0]
541     output_size = conv_shape[1]
542     input_size  = STRIDES[i] * output_size
543     conv = tf.reshape(conv, (batch_size, output_size,
        ↪  output_size, 3, 5 + NUM_CLASS))
544
```

```
545     conv_raw_conf = conv[:, :, :, :, 4:5]
546     conv_raw_prob = conv[:, :, :, :, 5:]
547 
548     pred_xywh     = pred[:, :, :, :, 0:4]
549     pred_conf     = pred[:, :, :, :, 4:5]
550 
551     label_xywh    = label[:, :, :, :, 0:4]
552     respond_bbox  = label[:, :, :, :, 4:5]
553     label_prob    = label[:, :, :, :, 5:]
554 
555     giou = tf.expand_dims(bbox_giou(pred_xywh,
        ↪  label_xywh), axis=-1)
556     input_size = tf.cast(input_size, tf.float32)
557 
558     bbox_loss_scale = 2.0 - 1.0 * label_xywh[:, :, :, :,
        ↪  2:3] * label_xywh[:, :, :, :, 3:4] / (input_size
        ↪  ** 2)
559     giou_loss = respond_bbox * bbox_loss_scale * (1 - giou)
560 
561     iou = bbox_iou(pred_xywh[:, :, :, :, np.newaxis, :],
        ↪  bboxes[:, np.newaxis, np.newaxis, np.newaxis, :,
        ↪  :])
562     # Find the value of IoU with the real box The largest
        ↪  prediction box
563     max_iou = tf.expand_dims(tf.reduce_max(iou, axis=-1),
        ↪  axis=-1)
564 
565     # If the largest iou is less than the threshold, it is
        ↪  considered that the prediction box contains no
        ↪  objects, then the background box
566     respond_bgd = (1.0 - respond_bbox) * tf.cast( max_iou
        ↪  < YOLO_IOU_LOSS_THRESH, tf.float32 )
567 
568     conf_focal = tf.pow(respond_bbox - pred_conf, 2)
569 
570     # Calculate the loss of confidence
571     # we hope that if the grid contains objects, then the
        ↪  network output prediction box has a confidence of
        ↪  1 and 0 when there is no object.
572     conf_loss = conf_focal * (
```

```
573             respond_bbox * tf.nn.sigmoid_cross_entropy_wit⌋
                ↪ h_logits(labels=respond_bbox,
                ↪ logits=conv_raw_conf)
574             +
575             respond_bgd * tf.nn.sigmoid_cross_entropy_with⌋
                ↪ _logits(labels=respond_bbox,
                ↪ logits=conv_raw_conf)
576     )
577
578     prob_loss = respond_bbox * tf.nn.sigmoid_cross_entropy⌋
        ↪ _with_logits(labels=label_prob,
        ↪ logits=conv_raw_prob)
579
580     giou_loss = tf.reduce_mean(tf.reduce_sum(giou_loss,
        ↪ axis=[1,2,3,4]))
581     conf_loss = tf.reduce_mean(tf.reduce_sum(conf_loss,
        ↪ axis=[1,2,3,4]))
582     prob_loss = tf.reduce_mean(tf.reduce_sum(prob_loss,
        ↪ axis=[1,2,3,4]))
583
584     return giou_loss, conf_loss, prob_loss
```

C.3.3 utils.py

```
#================================================================
#
#   File name   : utils.py
#   Author      : PyLessons
#   Created date: 2020-09-27
#   Website     : https://pylessons.com/
#   GitHub      : https://github.com/pythonlessons/TensorFlow-2.x-YOLOv3
#   Description : additional yolov3 and yolov4 functions
#   Modified by : Virginia Ramón
#================================================================
from multiprocessing import Process, Queue, Pipe
import cv2
import time
import random
import colorsys
import numpy as np
import tensorflow as tf
from yolov3.configs import *
from yolov3.yolov4 import *
from tensorflow.python.saved_model import tag_constants
#Added by Virginia Ramon
import json
import sys
sys.path.append("/home/virginia/TFG/YOLOv4/TensorFlow-2.x-YOLOv3-master/")
def load_yolo_weights(model, weights_file):
    tf.keras.backend.clear_session() # used to reset layer names
    # load Darknet original weights to TensorFlow model
    if YOLO_TYPE == "yolov3":
        range1 = 75 if not TRAIN_YOLO_TINY else 13
        range2 = [58, 66, 74] if not TRAIN_YOLO_TINY else [9, 12]
    if YOLO_TYPE == "yolov4":
        range1 = 110 if not TRAIN_YOLO_TINY else 21
```

```
33          range2 = [93, 101, 109] if not TRAIN_YOLO_TINY
             ↪  else [17, 20]
34
35      with open(weights_file, 'rb') as wf:
36          major, minor, revision, seen, _ = np.fromfile(wf,
             ↪  dtype=np.int32, count=5)
37
38          j = 0
39          for i in range(range1):
40              if i > 0:
41                  conv_layer_name = 'conv2d_%d' %i
42              else:
43                  conv_layer_name = 'conv2d'
44
45              if j > 0:
46                  bn_layer_name = 'batch_normalization_%d' %j
47              else:
48                  bn_layer_name = 'batch_normalization'
49
50              conv_layer = model.get_layer(conv_layer_name)
51              filters = conv_layer.filters
52              k_size = conv_layer.kernel_size[0]
53              in_dim = conv_layer.input_shape[-1]
54
55              if i not in range2:
56                  # darknet weights: [beta, gamma, mean,
                     ↪  variance]
57                  bn_weights = np.fromfile(wf,
                     ↪  dtype=np.float32, count=4 * filters)
58                  # tf weights: [gamma, beta, mean, variance]
59                  bn_weights = bn_weights.reshape((4,
                     ↪  filters))[[1, 0, 2, 3]]
60                  bn_layer = model.get_layer(bn_layer_name)
61                  j += 1
62              else:
63                  conv_bias = np.fromfile(wf,
                     ↪  dtype=np.float32, count=filters)
64
65              # darknet shape (out_dim, in_dim, height,
                 ↪  width)
66              conv_shape = (filters, in_dim, k_size, k_size)
```

```
67              conv_weights = np.fromfile(wf,
                ↪  dtype=np.float32,
                ↪  count=np.product(conv_shape))
68              # tf shape (height, width, in_dim, out_dim)
69              conv_weights = conv_weights.reshape(conv_shape
                ↪  ).transpose([2, 3, 1,
                ↪  0])
70
71              if i not in range2:
72                  conv_layer.set_weights([conv_weights])
73                  bn_layer.set_weights(bn_weights)
74              else:
75                  conv_layer.set_weights([conv_weights,
                    ↪  conv_bias])
76
77          assert len(wf.read()) == 0, 'failed to read all
            ↪  data'
78
79  def Load_Yolo_model():
80      physical_devices =
        ↪  tf.config.list_physical_devices('GPU')
81      for gpu_instance in physical_devices:
82          try: tf.config.experimental.set_memory_growth(gpu_
            ↪  instance,
            ↪  True)
83          except RuntimeError: pass
84
85      if YOLO_FRAMEWORK == "tf": # TensorFlow detection
86          if YOLO_TYPE == "yolov4":
87              Darknet_weights = YOLO_V4_TINY_WEIGHTS if
                ↪  TRAIN_YOLO_TINY else YOLO_V4_WEIGHTS
88          if YOLO_TYPE == "yolov3":
89              Darknet_weights = YOLO_V3_TINY_WEIGHTS if
                ↪  TRAIN_YOLO_TINY else YOLO_V3_WEIGHTS
90
91          if YOLO_CUSTOM_WEIGHTS == False:
92              print("Loading Darknet_weights from:",
                ↪  Darknet_weights)
93              yolo = Create_Yolo(input_size=YOLO_INPUT_SIZE,
                ↪  CLASSES=YOLO_COCO_CLASSES)
```

```
94              load_yolo_weights(yolo, Darknet_weights) # use
                ↪ Darknet weights
95          else:
96              checkpoint = f"/home/virginia/TFG/YOLOv4/Tenso
                ↪ rFlow-2.x-YOLOv3-master/checkpoints/yolov4
                ↪ _PigPersv5/{TRAIN_MODEL_NAME}"
97              if TRAIN_YOLO_TINY:
98                  checkpoint += "_Tiny"
99              print("Loading custom weights from:",
                ↪ checkpoint)
100             yolo = Create_Yolo(input_size=YOLO_INPUT_SIZE,
                ↪ CLASSES=TRAIN_CLASSES)
101             yolo.load_weights(checkpoint)  # use custom
                ↪ weights
102
103     elif YOLO_FRAMEWORK == "trt": # TensorRT detection
104         saved_model_loaded =
            ↪ tf.saved_model.load(YOLO_CUSTOM_WEIGHTS,
            ↪ tags=[tag_constants.SERVING])
105         signature_keys =
            ↪ list(saved_model_loaded.signatures.keys())
106         yolo = saved_model_loaded.signatures['serving_defa
            ↪ ult']
107
108     return yolo
109
110 def image_preprocess(image, target_size, gt_boxes=None):
111     ih, iw    = target_size
112     h,  w, _  = image.shape
113
114     scale = min(iw/w, ih/h)
115     nw, nh  = int(scale * w), int(scale * h)
116     image_resized = cv2.resize(image, (nw, nh))
117
118     image_paded = np.full(shape=[ih, iw, 3],
        ↪ fill_value=128.0)
119     dw, dh = (iw - nw) // 2, (ih-nh) // 2
120     image_paded[dh:nh+dh, dw:nw+dw, :] = image_resized
121     image_paded = image_paded / 255.
122
123     if gt_boxes is None:
```

```
124         return image_paded
125
126     else:
127         gt_boxes[:, [0, 2]] = gt_boxes[:, [0, 2]] * scale
            ↪  + dw
128         gt_boxes[:, [1, 3]] = gt_boxes[:, [1, 3]] * scale
            ↪  + dh
129         return image_paded, gt_boxes
130
131
132 def draw_bbox(image, bboxes, CLASSES=YOLO_COCO_CLASSES,
    ↪  show_label=True, show_confidence = True,
    ↪  Text_colors=(255,255,0), rectangle_colors='',
    ↪  tracking=False):
133     NUM_CLASS = read_class_names(CLASSES)
134     num_classes = len(NUM_CLASS)
135     image_h, image_w, _ = image.shape
136     hsv_tuples = [(1.0 * x / num_classes, 1., 1.) for x in
        ↪  range(num_classes)]
137     #print("hsv_tuples", hsv_tuples)
138     colors = list(map(lambda x: colorsys.hsv_to_rgb(*x),
        ↪  hsv_tuples))
139     colors = list(map(lambda x: (int(x[0] * 255), int(x[1]
        ↪  * 255), int(x[2] * 255)), colors))
140
141     random.seed(0)
142     random.shuffle(colors)
143     random.seed(None)
144
145     for i, bbox in enumerate(bboxes):
146         coor = np.array(bbox[:4], dtype=np.int32)
147         score = bbox[4]
148         class_ind = int(bbox[5])
149         bbox_color = rectangle_colors if rectangle_colors
            ↪  != '' else colors[class_ind]
150         bbox_thick = int(0.6 * (image_h + image_w) / 1300)
151         if bbox_thick < 1: bbox_thick = 1
152         fontScale = 0.5 * bbox_thick
153         (x1, y1), (x2, y2) = (coor[0], coor[1]), (coor[2],
            ↪  coor[3])
154
```

```
155         # put object rectangle
156         cv2.rectangle(image, (x1, y1), (x2, y2),
             ↪  bbox_color, bbox_thick*2)
157
158         if show_label:
159             # get text label
160             score_str = " {:.2f}".format(score) if
                 ↪  show_confidence else ""
161
162             if tracking: score_str = " "+str(score)
163
164             try:
165                 label = "{}".format(NUM_CLASS[class_ind])
                     ↪  + score_str
166             except KeyError:
167                 print("You received KeyError, this might
                     ↪  be that you are trying to use yolo
                     ↪  original weights")
168                 print("while using custom classes, if
                     ↪  using custom model in configs.py set
                     ↪  YOLO_CUSTOM_WEIGHTS = True")
169
170             # get text size
171             (text_width, text_height), baseline =
                 ↪  cv2.getTextSize(label,
                 ↪  cv2.FONT_HERSHEY_COMPLEX_SMALL,
```

```
fontScale , thickness = bbox-
```

```
173             # put filled text rectangle
174             cv2.rectangle(image, (x1, y1), (x1 +
                ↪ text_width, y1 - text_height - baseline),
                ↪ bbox_color, thickness=cv2.FILLED)
175
176             # put text above rectangle
177             cv2.putText(image, label, (x1, y1-4),
                ↪ cv2.FONT_HERSHEY_COMPLEX_SMALL,
178                         fontScale, Text_colors,
                            ↪ bbox_thick,
                            ↪ lineType=cv2.LINE_AA)
179
180     return image
181
182
183 def bboxes_iou(boxes1, boxes2):
184     boxes1 = np.array(boxes1)
185     boxes2 = np.array(boxes2)
186
187     boxes1_area = (boxes1[..., 2] - boxes1[..., 0]) *
        ↪ (boxes1[..., 3] - boxes1[..., 1])
188     boxes2_area = (boxes2[..., 2] - boxes2[..., 0]) *
        ↪ (boxes2[..., 3] - boxes2[..., 1])
189
190     left_up       = np.maximum(boxes1[..., :2],
        ↪ boxes2[..., :2])
191     right_down    = np.minimum(boxes1[..., 2:],
        ↪ boxes2[..., 2:])
192
193     inter_section = np.maximum(right_down - left_up, 0.0)
194     inter_area    = inter_section[..., 0] *
        ↪ inter_section[..., 1]
195     union_area    = boxes1_area + boxes2_area - inter_area
196     ious          = np.maximum(1.0 * inter_area /
        ↪ union_area, np.finfo(np.float32).eps)
197
198     return ious
199
200
201 def nms(bboxes, iou_threshold, sigma=0.3, method='nms'):
202     """
```

```
    :param bboxes: (xmin, ymin, xmax, ymax, score, class)

    Note: soft-nms, https://arxiv.org/pdf/1704.04503.pdf
          https://github.com/bharatsingh430/soft-nms
    """
    classes_in_img = list(set(bboxes[:, 5]))
    best_bboxes = []

    for cls in classes_in_img:
        cls_mask = (bboxes[:, 5] == cls)
        cls_bboxes = bboxes[cls_mask]
        # Process 1: Determine whether the number of
         ↪ bounding boxes is greater than 0
        while len(cls_bboxes) > 0:
            # Process 2: Select the bounding box with the
             ↪ highest score according to socre order A
            max_ind = np.argmax(cls_bboxes[:, 4])
            best_bbox = cls_bboxes[max_ind]
            best_bboxes.append(best_bbox)
            cls_bboxes = np.concatenate([cls_bboxes[:
             ↪ max_ind], cls_bboxes[max_ind + 1:]])
            # Process 3: Calculate this bounding box A and
            # Remain all iou of the bounding box and
             ↪ remove those bounding boxes whose iou
             ↪ value is higher than the threshold
            iou = bboxes_iou(best_bbox[np.newaxis, :4],
             ↪ cls_bboxes[:, :4])
            weight = np.ones((len(iou),), dtype=np.float32)

            assert method in ['nms', 'soft-nms']

            if method == 'nms':
                iou_mask = iou > iou_threshold
                weight[iou_mask] = 0.0

            if method == 'soft-nms':
                weight = np.exp(-(1.0 * iou ** 2 / sigma))

            cls_bboxes[:, 4] = cls_bboxes[:, 4] * weight
            score_mask = cls_bboxes[:, 4] > 0.
            cls_bboxes = cls_bboxes[score_mask]
```

```
238
239     return best_bboxes
240
241
242 def postprocess_boxes(pred_bbox, original_image,
    ↪ input_size, score_threshold):
243     valid_scale=[0, np.inf]
244     pred_bbox = np.array(pred_bbox)
245
246     pred_xywh = pred_bbox[:, 0:4]
247     pred_conf = pred_bbox[:, 4]
248     pred_prob = pred_bbox[:, 5:]
249
250     # 1. (x, y, w, h) --> (xmin, ymin, xmax, ymax)
251     pred_coor = np.concatenate([pred_xywh[:, :2] -
        ↪ pred_xywh[:, 2:] * 0.5,
252                                 pred_xywh[:, :2] +
                                    ↪ pred_xywh[:, 2:] *
                                    ↪ 0.5], axis=-1)
253     # 2. (xmin, ymin, xmax, ymax) -> (xmin_org, ymin_org,
        ↪ xmax_org, ymax_org)
254     org_h, org_w = original_image.shape[:2]
255     resize_ratio = min(input_size / org_w, input_size /
        ↪ org_h)
256
257     dw = (input_size - resize_ratio * org_w) / 2
258     dh = (input_size - resize_ratio * org_h) / 2
259
260     pred_coor[:, 0::2] = 1.0 * (pred_coor[:, 0::2] - dw) /
        ↪ resize_ratio
261     pred_coor[:, 1::2] = 1.0 * (pred_coor[:, 1::2] - dh) /
        ↪ resize_ratio
262
263     # 3. clip some boxes those are out of range
264     pred_coor = np.concatenate([np.maximum(pred_coor[:,
        ↪ :2], [0, 0]),
265                                 np.minimum(pred_coor[:,
                                    ↪ 2:], [org_w - 1, org_h
                                    ↪ - 1])], axis=-1)
```

```
266     invalid_mask = np.logical_or((pred_coor[:, 0] >
        ↪ pred_coor[:, 2]), (pred_coor[:, 1] > pred_coor[:,
        ↪ 3]))
267     pred_coor[invalid_mask] = 0
268
269     # 4. discard some invalid boxes
270     bboxes_scale = np.sqrt(np.multiply.reduce(pred_coor[:,
        ↪ 2:4] - pred_coor[:, 0:2], axis=-1))
271     scale_mask = np.logical_and((valid_scale[0] <
        ↪ bboxes_scale), (bboxes_scale < valid_scale[1]))
272
273     # 5. discard boxes with low scores
274     classes = np.argmax(pred_prob, axis=-1)
275     scores = pred_conf *
        ↪ pred_prob[np.arange(len(pred_coor)), classes]
276     score_mask = scores > score_threshold
277     mask = np.logical_and(scale_mask, score_mask)
278     coors, scores, classes = pred_coor[mask],
        ↪ scores[mask], classes[mask]
279
280     return np.concatenate([coors, scores[:, np.newaxis],
        ↪ classes[:, np.newaxis]], axis=-1)
281
282
283 def detect_image(Yolo, image_path, output_path,
    ↪ input_size=416, show=False, CLASSES=YOLO_COCO_CLASSES,
    ↪ score_threshold=0.5, iou_threshold=0.45,
    ↪ rectangle_colors=''):
284     original_image      = cv2.imread(image_path)
285     original_image      = cv2.cvtColor(original_image,
        ↪ cv2.COLOR_BGR2RGB)
286     original_image      = cv2.cvtColor(original_image,
        ↪ cv2.COLOR_BGR2RGB)
287
288     image_data = image_preprocess(np.copy(original_image),
        ↪ [input_size, input_size])
289     image_data = image_data[np.newaxis,
        ↪ ...].astype(np.float32)
290
291     if YOLO_FRAMEWORK == "tf":
292         pred_bbox = Yolo.predict(image_data)
```

```
293     elif YOLO_FRAMEWORK == "trt":
294         batched_input = tf.constant(image_data)
295         result = Yolo(batched_input)
296         pred_bbox = []
297         for key, value in result.items():
298             value = value.numpy()
299             pred_bbox.append(value)
300 
301     pred_bbox = [tf.reshape(x, (-1, tf.shape(x)[-1])) for
        ↪ x in pred_bbox]
302     pred_bbox = tf.concat(pred_bbox, axis=0)
303 
304     bboxes = postprocess_boxes(pred_bbox, original_image,
        ↪ input_size, score_threshold)
305     bboxes = nms(bboxes, iou_threshold, method='nms')
306 
307     image = draw_bbox(original_image, bboxes,
        ↪ CLASSES=CLASSES, rectangle_colors=rectangle_colors)
308     #--Added by Virginia Ramon--
309     data = [output_path, bboxes]
310 
311     #print("Detections: "+ str(int(time.time())) +" "+
        ↪ image_path)
312     #for b in bboxes:
313     #    print(b)
314     #print(read_class_names(CLASSES))
315     #--------------------------
316     if output_path != '': cv2.imwrite(output_path, image)
317     if show:
318         # Show the image
319         cv2.imshow("predicted image", image)
320         # Load and hold the image
321         cv2.waitKey(0)
322         # To close the window after the required kill
            ↪ value was provided
323         cv2.destroyAllWindows()
324 
325     return image, data, read_class_names(CLASSES)
326 
327 #--Added by Virginia Ramon--
```

```
328 def detect_frame(Yolo, original_image, input_size=416,
    ↪ show=False, CLASSES=YOLO_COCO_CLASSES,
    ↪ score_threshold=0.5, iou_threshold=0.45):
329     original_image      = cv2.cvtColor(original_image,
        ↪ cv2.COLOR_BGR2RGB)
330     original_image      = cv2.cvtColor(original_image,
        ↪ cv2.COLOR_BGR2RGB)
331
332     image_data = image_preprocess(np.copy(original_image),
        ↪ [input_size, input_size])
333     image_data = image_data[np.newaxis,
        ↪ ...].astype(np.float32)
334
335     if YOLO_FRAMEWORK == "tf":
336         pred_bbox = Yolo.predict(image_data)
337     elif YOLO_FRAMEWORK == "trt":
338         batched_input = tf.constant(image_data)
339         result = Yolo(batched_input)
340         pred_bbox = []
341         for key, value in result.items():
342             value = value.numpy()
343             pred_bbox.append(value)
344
345     pred_bbox = [tf.reshape(x, (-1, tf.shape(x)[-1])) for
        ↪ x in pred_bbox]
346     pred_bbox = tf.concat(pred_bbox, axis=0)
347
348     bboxes = postprocess_boxes(pred_bbox, original_image,
        ↪ input_size, score_threshold)
349     bboxes = nms(bboxes, iou_threshold, method='nms')
350
351     #image = draw_bbox(original_image, bboxes,
        ↪ CLASSES=CLASSES, rectangle_colors=rectangle_colors)
352
353     return bboxes, read_class_names(CLASSES)
354 #-------------------------
355
356 def Predict_bbox_mp(Frames_data, Predicted_data,
    ↪ Processing_times):
357     gpus =
        ↪ tf.config.experimental.list_physical_devices('GPU')
```

```
358     if len(gpus) > 0:
359         try: tf.config.experimental.set_memory_growth(gpus[0],
            ↪ True)
360         except RuntimeError: print("RuntimeError in tf.config.experimental.list_physical_devices('GPU')")
361     Yolo = Load_Yolo_model()
362     times = []
363     while True:
364         if Frames_data.qsize()>0:
365             image_data = Frames_data.get()
366             t1 = time.time()
367             Processing_times.put(time.time())
368 
369             if YOLO_FRAMEWORK == "tf":
370                 pred_bbox = Yolo.predict(image_data)
371             elif YOLO_FRAMEWORK == "trt":
372                 batched_input = tf.constant(image_data)
373                 result = Yolo(batched_input)
374                 pred_bbox = []
375                 for key, value in result.items():
376                     value = value.numpy()
377                     pred_bbox.append(value)
378 
379             pred_bbox = [tf.reshape(x, (-1,
                ↪ tf.shape(x)[-1])) for x in pred_bbox]
380             pred_bbox = tf.concat(pred_bbox, axis=0)
381 
382             Predicted_data.put(pred_bbox)
383 
384 
385 def postprocess_mp(Predicted_data, original_frames,
    ↪ Processed_frames, Processing_times, input_size,
    ↪ CLASSES, score_threshold, iou_threshold,
    ↪ rectangle_colors, realtime):
386     times = []
387     while True:
388         if Predicted_data.qsize()>0:
389             pred_bbox = Predicted_data.get()
390             if realtime:
391                 while original_frames.qsize() > 1:
```

```
392                     original_image = original_frames.get()
393             else:
394                 original_image = original_frames.get()
395 
396             bboxes = postprocess_boxes(pred_bbox,
                ↪ original_image, input_size,
                ↪ score_threshold)
397             bboxes = nms(bboxes, iou_threshold,
                ↪ method='nms')
398             image = draw_bbox(original_image, bboxes,
                ↪ CLASSES=CLASSES,
                ↪ rectangle_colors=rectangle_colors)
399             times.append(time.time()-Processing_times.get(」
                ↪ ))
400             times = times[-20:]
401 
402             ms = sum(times)/len(times)*1000
403             fps = 1000 / ms
404             image = cv2.putText(image, "Time:
                ↪ {:.1f}FPS".format(fps), (0, 30),
                ↪ cv2.FONT_HERSHEY_COMPLEX_SMALL, 1, (0, 0,
                ↪ 255), 2)
405             #print("Time: {:.2f}ms, Final FPS:
                ↪ {:.1f}".format(ms, fps))
406 
407             Processed_frames.put(image)
408 
409 def Show_Image_mp(Processed_frames, show, Final_frames):
410     while True:
411         if Processed_frames.qsize()>0:
412             image = Processed_frames.get()
413             Final_frames.put(image)
414             if show:
415                 cv2.imshow('output', image)
416                 if cv2.waitKey(25) & 0xFF == ord("q"):
417                     cv2.destroyAllWindows()
418                     break
419 
420 # detect from webcam
```

```
421 def detect_video_realtime_mp(video_path, output_path,
    ↪ input_size=416, show=False, CLASSES=YOLO_COCO_CLASSES,
    ↪ score_threshold=0.3, iou_threshold=0.45,
    ↪ rectangle_colors='', realtime=False):
422     if realtime:
423         vid = cv2.VideoCapture(0)
424     else:
425         vid = cv2.VideoCapture(video_path)
426 
427     # by default VideoCapture returns float instead of int
428     width = int(vid.get(cv2.CAP_PROP_FRAME_WIDTH))
429     height = int(vid.get(cv2.CAP_PROP_FRAME_HEIGHT))
430     fps = int(vid.get(cv2.CAP_PROP_FPS))
431     codec = cv2.VideoWriter_fourcc(*'XVID')
432     out = cv2.VideoWriter(output_path, codec, fps, (width,
        ↪ height)) # output_path must be .mp4
433     no_of_frames = int(vid.get(cv2.CAP_PROP_FRAME_COUNT))
434 
435     original_frames = Queue()
436     Frames_data = Queue()
437     Predicted_data = Queue()
438     Processed_frames = Queue()
439     Processing_times = Queue()
440     Final_frames = Queue()
441 
442     p1 = Process(target=Predict_bbox_mp,
        ↪ args=(Frames_data, Predicted_data,
        ↪ Processing_times))
443     p2 = Process(target=postprocess_mp,
        ↪ args=(Predicted_data, original_frames,
        ↪ Processed_frames, Processing_times, input_size,
        ↪ CLASSES, score_threshold, iou_threshold,
        ↪ rectangle_colors, realtime))
444     p3 = Process(target=Show_Image_mp,
        ↪ args=(Processed_frames, show, Final_frames))
445     p1.start()
446     p2.start()
447     p3.start()
448 
449     while True:
450         ret, img = vid.read()
```

```
451             if not ret:
452                 break
453 
454             original_image = cv2.cvtColor(img,
                ↪  cv2.COLOR_BGR2RGB)
455             original_image = cv2.cvtColor(original_image,
                ↪  cv2.COLOR_BGR2RGB)
456             original_frames.put(original_image)
457 
458             image_data =
                ↪  image_preprocess(np.copy(original_image),
                ↪  [input_size, input_size])
459             image_data = image_data[np.newaxis,
                ↪  ...].astype(np.float32)
460             Frames_data.put(image_data)
461 
462         while True:
463             if original_frames.qsize() == 0 and
                ↪  Frames_data.qsize() == 0 and
                ↪  Predicted_data.qsize() == 0  and
                ↪  Processed_frames.qsize() == 0  and
                ↪  Processing_times.qsize() == 0 and
                ↪  Final_frames.qsize() == 0:
464                 p1.terminate()
465                 p2.terminate()
466                 p3.terminate()
467                 break
468             elif Final_frames.qsize()>0:
469                 image = Final_frames.get()
470                 if output_path != '': out.write(image)
471 
472         cv2.destroyAllWindows()
473 
474 def detect_video(Yolo, video_path, output_path,
    ↪  input_size=416, show=False, CLASSES=YOLO_COCO_CLASSES,
    ↪  score_threshold=0.5, iou_threshold=0.45,
    ↪  rectangle_colors=''):
475     times, times_2 = [], []
476     vid = cv2.VideoCapture(video_path)
477     #--Added by Virginia Ramon--
478     data = []
```

```
479     frame = 0
480     #--------------------------
481 
482     # by default VideoCapture returns float instead of int
483     width = int(vid.get(cv2.CAP_PROP_FRAME_WIDTH))
484     height = int(vid.get(cv2.CAP_PROP_FRAME_HEIGHT))
485     fps = int(vid.get(cv2.CAP_PROP_FPS))
486     codec = cv2.VideoWriter_fourcc(*'XVID')
487     out = cv2.VideoWriter(output_path, codec, fps, (width,
        ↪  height)) # output_path must be .mp4
488 
489     while True:
490         _, img = vid.read()
491 
492         try:
493             original_image = cv2.cvtColor(img,
                ↪  cv2.COLOR_BGR2RGB)
494             original_image = cv2.cvtColor(original_image,
                ↪  cv2.COLOR_BGR2RGB)
495         except:
496             break
497 
498         image_data =
            ↪  image_preprocess(np.copy(original_image),
            ↪  [input_size, input_size])
499         image_data = image_data[np.newaxis,
            ↪  ...].astype(np.float32)
500 
501         t1 = time.time()
502         if YOLO_FRAMEWORK == "tf":
503             pred_bbox = Yolo.predict(image_data)
504         elif YOLO_FRAMEWORK == "trt":
505             batched_input = tf.constant(image_data)
506             result = Yolo(batched_input)
507             pred_bbox = []
508             for key, value in result.items():
509                 value = value.numpy()
510                 pred_bbox.append(value)
511 
512         t2 = time.time()
513 
```

```
514         pred_bbox = [tf.reshape(x, (-1, tf.shape(x)[-1]))
             ↪  for x in pred_bbox]
515         pred_bbox = tf.concat(pred_bbox, axis=0)
516
517         bboxes = postprocess_boxes(pred_bbox,
             ↪  original_image, input_size, score_threshold)
518         bboxes = nms(bboxes, iou_threshold, method='nms')
519
520         image = draw_bbox(original_image, bboxes,
             ↪  CLASSES=CLASSES,
             ↪  rectangle_colors=rectangle_colors)
521
522         t3 = time.time()
523         times.append(t2-t1)
524         times_2.append(t3-t1)
525
526         times = times[-20:]
527         times_2 = times_2[-20:]
528
529         ms = sum(times)/len(times)*1000
530         fps = 1000 / ms
531         fps2 = 1000 / (sum(times_2)/len(times_2)*1000)
532
533         image = cv2.putText(image, "Time:
             ↪  {:.1f}FPS".format(fps), (0, 30),
             ↪  cv2.FONT_HERSHEY_COMPLEX_SMALL, 1, (0, 0,
             ↪  255), 2)
534         #CreateXMLfile("XML_Detections",
             ↪  str(int(time.time())), original_image, bboxes,
             ↪  read_class_names(CLASSES))
535
536         #--Added by Virginia Ramon--
537         data.append([video_path, frame, bboxes])
538         frame += 1
539         #-------------------------
540
541         print("Time: {:.2f}ms, Detection FPS: {:.1f},
             ↪  total FPS: {:.1f}".format(ms, fps, fps2))
542         if output_path != '': out.write(image)
543         if show:
544             cv2.imshow('output', image)
```

```
545             if cv2.waitKey(25) & 0xFF == ord("q"):
546                 cv2.destroyAllWindows()
547                 break
548 
549     cv2.destroyAllWindows()
550     return data, read_class_names(CLASSES)
551 
552 # detect from webcam
553 def detect_realtime(Yolo, output_path, input_size=416,
    ↪  show=False, CLASSES=YOLO_COCO_CLASSES,
    ↪  score_threshold=0.3, iou_threshold=0.45,
    ↪  rectangle_colors=''):
554     times = []
555     vid = cv2.VideoCapture(0)
556 
557     # by default VideoCapture returns float instead of int
558     width = int(vid.get(cv2.CAP_PROP_FRAME_WIDTH))
559     height = int(vid.get(cv2.CAP_PROP_FRAME_HEIGHT))
560     fps = int(vid.get(cv2.CAP_PROP_FPS))
561     codec = cv2.VideoWriter_fourcc(*'XVID')
562     out = cv2.VideoWriter(output_path, codec, fps, (width,
        ↪  height)) # output_path must be .mp4
563 
564     while True:
565         _, frame = vid.read()
566 
567         try:
568             original_frame = cv2.cvtColor(frame,
                ↪  cv2.COLOR_BGR2RGB)
569             original_frame = cv2.cvtColor(original_frame,
                ↪  cv2.COLOR_BGR2RGB)
570         except:
571             break
572         image_data =
            ↪  image_preprocess(np.copy(original_frame),
            ↪  [input_size, input_size])
573         image_data = image_data[np.newaxis,
            ↪  ...].astype(np.float32)
574 
575         t1 = time.time()
576         if YOLO_FRAMEWORK == "tf":
```

```
577             pred_bbox = Yolo.predict(image_data)
578         elif YOLO_FRAMEWORK == "trt":
579             batched_input = tf.constant(image_data)
580             result = Yolo(batched_input)
581             pred_bbox = []
582             for key, value in result.items():
583                 value = value.numpy()
584                 pred_bbox.append(value)
585 
586         t2 = time.time()
587 
588         pred_bbox = [tf.reshape(x, (-1, tf.shape(x)[-1]))
             ↪  for x in pred_bbox]
589         pred_bbox = tf.concat(pred_bbox, axis=0)
590 
591         bboxes = postprocess_boxes(pred_bbox,
             ↪  original_frame, input_size, score_threshold)
592         bboxes = nms(bboxes, iou_threshold, method='nms')
593 
594         times.append(t2-t1)
595         times = times[-20:]
596 
597         ms = sum(times)/len(times)*1000
598         fps = 1000 / ms
599 
600         print("Time: {:.2f}ms, {:.1f} FPS".format(ms, fps))
601 
602         frame = draw_bbox(original_frame, bboxes,
             ↪  CLASSES=CLASSES,
             ↪  rectangle_colors=rectangle_colors)
603         # CreateXMLfile("XML_Detections",
             ↪  str(int(time.time())), original_frame, bboxes,
             ↪  read_class_names(CLASSES))
604         image = cv2.putText(frame, "Time:
             ↪  {:.1f}FPS".format(fps), (0, 30),
605                             cv2.FONT_HERSHEY_COMPLEX_SMALL,
                                 ↪  1, (0, 0, 255), 2)
606 
607         if output_path != '': out.write(frame)
608         if show:
609             cv2.imshow('output', frame)
```

```
610             if cv2.waitKey(25) & 0xFF == ord("q"):
611                 cv2.destroyAllWindows()
612                 break
613 
614     cv2.destroyAllWindows()
```

C.3.4 dataset.py

```
#=============================================================
#
#   File name   : dataset.py
#   Author      : PyLessons
#   Created date: 2020-07-31
#   Website     : https://pylessons.com/
#   GitHub      : https://github.com/pythonlessons/TensorFlow-2.x-YOLOv3
#   Description : functions used to prepare dataset for custom training
#
#=============================================================
# TODO: transfer numpy to tensorflow operations
import os
import sys
import cv2
import random
import numpy as np
import tensorflow as tf
from yolov3.utils import read_class_names, image_preprocess
from yolov3.yolov3 import bbox_iou
from yolov3.configs import *


class Dataset(object):
    # Dataset preprocess implementation
    def __init__(self, dataset_type, TEST_INPUT_SIZE=TEST_INPUT_SIZE):
        self.annot_path  = TRAIN_ANNOT_PATH if dataset_type == 'train' else TEST_ANNOT_PATH
        self.input_sizes = TRAIN_INPUT_SIZE if dataset_type == 'train' else TEST_INPUT_SIZE
        self.batch_size  = TRAIN_BATCH_SIZE if dataset_type == 'train' else TEST_BATCH_SIZE
        self.data_aug    = TRAIN_DATA_AUG   if dataset_type == 'train' else TEST_DATA_AUG

```

```
31          self.train_yolo_tiny = TRAIN_YOLO_TINY
32          self.train_input_sizes = TRAIN_INPUT_SIZE
33          self.strides = np.array(YOLO_STRIDES)
34          self.classes = read_class_names(TRAIN_CLASSES)
35          self.num_classes = len(self.classes)
36          self.anchors =
             ↪  (np.array(YOLO_ANCHORS).T/self.strides).T
37          self.anchor_per_scale = YOLO_ANCHOR_PER_SCALE
38          self.max_bbox_per_scale = YOLO_MAX_BBOX_PER_SCALE
39
40          self.annotations =
             ↪  self.load_annotations(dataset_type)
41          self.num_samples = len(self.annotations)
42          self.num_batchs = int(np.ceil(self.num_samples /
             ↪  self.batch_size))
43          self.batch_count = 0
44
45
46      def load_annotations(self, dataset_type):
47          final_annotations = []
48          with open(self.annot_path, 'r') as f:
49              txt = f.read().splitlines()
50              annotations = [line.strip() for line in txt if
                 ↪  len(line.strip().split()[1:]) != 0]
51          np.random.shuffle(annotations)
52          n = 1
53          # for annotation in annotations:
54          #     image_extension = '.jpg'
55          #     extension_index =
             ↪  annotation.find(image_extension)
56          #     image_path = annotation[:extension_index+len
             ↪  (image_extension)]
57          #     line = annotation[extension_index+len(image_
             ↪  extension):].split()
58          #     if not os.path.exists(image_path):
59          #         raise KeyError("%s does not exist ... "
             ↪  %image_path)
60          #     if TRAIN_LOAD_IMAGES_TO_RAM:
61          #         image = cv2.imread(image_path)
62          #     else:
63          #         image = ''
```

```
64          #     final_annotations.append([image_path, line,
             ↪  image])
65          # return final_annotations
66          for annotation in annotations:
67              if n%100 == 0:
68                  print(str(n) + '/' + str(len(annotations)),
                     ↪  flush = True)
69              #sys.stdout.flush()
70              n += 1
71              # fully parse annotations
72              line = annotation.split()
73              image_path, index = "", 1
74              for i, one_line in enumerate(line):
75                  if not
                     ↪  one_line.replace(",","").isnumeric():
76                      if image_path != "": image_path += " "
77                      image_path += one_line
78                  else:
79                      index = i
80                      break
81              if not os.path.exists(image_path):
82                  raise KeyError("%s does not exist ... "
                     ↪  %image_path)
83              if TRAIN_LOAD_IMAGES_TO_RAM:
84                  image = cv2.imread(image_path)
85              else:
86                  image = ''
87              final_annotations.append([image_path,
                 ↪  line[index:], image])
88          return final_annotations
89
90      def __iter__(self):
91          return self
92
93      def Delete_bad_annotation(self, bad_annotation):
94          print(f'Deleting {bad_annotation} annotation line')
95          bad_image_path = bad_annotation[0]
96          bad_image_name = bad_annotation[0].split('/')[-1] #
             ↪  can be used to delete bad image
97          bad_xml_path = bad_annotation[0][:-3]+'xml' # can
             ↪  be used to delete bad xml file
```

```
98
99          # remove bad annotation line from annotation file
100         with open(self.annot_path, "r+") as f:
101             d = f.readlines()
102             f.seek(0)
103             for i in d:
104                 if bad_image_name not in i:
105                     f.write(i)
106             f.truncate()
107
108     def __next__(self):
109         with tf.device('/cpu:0'):
110             self.train_input_size =
                ↪ random.choice([self.train_input_sizes])
111             self.train_output_sizes =
                ↪ self.train_input_size // self.strides
112
113             batch_image = np.zeros((self.batch_size,
                ↪ self.train_input_size,
                ↪ self.train_input_size, 3),
                ↪ dtype=np.float32)
114
115             if self.train_yolo_tiny:
116                 batch_label_mbbox =
                    ↪ np.zeros((self.batch_size,
                    ↪ self.train_output_sizes[0],
                    ↪ self.train_output_sizes[0],
                    ↪ self.anchor_per_scale, 5 +
                    ↪ self.num_classes), dtype=np.float32)
117                 batch_label_lbbox =
                    ↪ np.zeros((self.batch_size,
                    ↪ self.train_output_sizes[1],
                    ↪ self.train_output_sizes[1],
                    ↪ self.anchor_per_scale, 5 +
                    ↪ self.num_classes), dtype=np.float32)
118             else:
```

```
119                 batch_label_sbbox =
                     ↪  np.zeros((self.batch_size,
                     ↪  self.train_output_sizes[0],
                     ↪  self.train_output_sizes[0],
                     ↪  self.anchor_per_scale, 5 +
                     ↪  self.num_classes), dtype=np.float32)
120                 batch_label_mbbox =
                     ↪  np.zeros((self.batch_size,
                     ↪  self.train_output_sizes[1],
                     ↪  self.train_output_sizes[1],
                     ↪  self.anchor_per_scale, 5 +
                     ↪  self.num_classes), dtype=np.float32)
121                 batch_label_lbbox =
                     ↪  np.zeros((self.batch_size,
                     ↪  self.train_output_sizes[2],
                     ↪  self.train_output_sizes[2],
                     ↪  self.anchor_per_scale, 5 +
                     ↪  self.num_classes), dtype=np.float32)
122
123                 batch_sbboxes = np.zeros((self.batch_size,
                     ↪  self.max_bbox_per_scale, 4),
                     ↪  dtype=np.float32)
124
125             batch_mbboxes = np.zeros((self.batch_size,
                 ↪  self.max_bbox_per_scale, 4),
                 ↪  dtype=np.float32)
126             batch_lbboxes = np.zeros((self.batch_size,
                 ↪  self.max_bbox_per_scale, 4),
                 ↪  dtype=np.float32)
127
128             exceptions = False
129             num = 0
130             if self.batch_count < self.num_batchs:
131                 while num < self.batch_size:
132                     index = self.batch_count *
                         ↪  self.batch_size + num
133                     if index >= self.num_samples: index -=
                         ↪  self.num_samples
134                     annotation = self.annotations[index]
135                     image, bboxes =
                         ↪  self.parse_annotation(annotation)
```

```
136                     try:
137                         if self.train_yolo_tiny:
138                             label_mbbox, label_lbbox,
                                ↪ mbboxes, lbboxes = self.pr
                                ↪ eprocess_true_boxes(bboxes)
139                         else:
140                             label_sbbox, label_mbbox,
                                ↪ label_lbbox, sbboxes,
                                ↪ mbboxes, lbboxes = self.pr
                                ↪ eprocess_true_boxes(bboxes)
141                     except IndexError:
142                         exceptions = True
143                         self.Delete_bad_annotation(annotat
                            ↪ ion)
144                         print("IndexError, something wrong
                            ↪ with", annotation[0], "removed
                            ↪ this line from annotation
                            ↪ file")
145
146                     batch_image[num, :, :, :] = image
147                     batch_label_mbbox[num, :, :, :, :] =
                        ↪ label_mbbox
148                     batch_label_lbbox[num, :, :, :, :] =
                        ↪ label_lbbox
149                     batch_mbboxes[num, :, :] = mbboxes
150                     batch_lbboxes[num, :, :] = lbboxes
151                     if not self.train_yolo_tiny:
152                         batch_label_sbbox[num, :, :, :, :]
                            ↪ = label_sbbox
153                         batch_sbboxes[num, :, :] = sbboxes
154
155                     num += 1
156
157                 if exceptions:
158                     print('\n')
159                     raise Exception("There were problems
                        ↪ with dataset, I fixed them, now
                        ↪ restart the training process.")
160                 self.batch_count += 1
161                 if not self.train_yolo_tiny:
```

```
                    batch_smaller_target =
                     ↪ batch_label_sbbox, batch_sbboxes
                batch_medium_target  = batch_label_mbbox,
                 ↪ batch_mbboxes
                batch_larger_target  = batch_label_lbbox,
                 ↪ batch_lbboxes

                if self.train_yolo_tiny:
                    return batch_image,
                     ↪ (batch_medium_target,
                     ↪ batch_larger_target)
                return batch_image, (batch_smaller_target,
                 ↪ batch_medium_target,
                 ↪ batch_larger_target)
            else:
                self.batch_count = 0
                np.random.shuffle(self.annotations)
                raise StopIteration

    def random_horizontal_flip(self, image, bboxes):
        if random.random() < 0.5:
            _, w, _ = image.shape
            image = image[:, ::-1, :]
            bboxes[:, [0,2]] = w - bboxes[:, [2,0]]

        return image, bboxes

    def random_crop(self, image, bboxes):
        if random.random() < 0.5:
            h, w, _ = image.shape
            max_bbox = np.concatenate([np.min(bboxes[:,
             ↪ 0:2], axis=0), np.max(bboxes[:, 2:4],
             ↪ axis=0)], axis=-1)

            max_l_trans = max_bbox[0]
            max_u_trans = max_bbox[1]
            max_r_trans = w - max_bbox[2]
            max_d_trans = h - max_bbox[3]

            crop_xmin = max(0, int(max_bbox[0] -
             ↪ random.uniform(0, max_l_trans)))
```

```
193             crop_ymin = max(0, int(max_bbox[1] -
                ↪  random.uniform(0, max_u_trans)))
194             crop_xmax = max(w, int(max_bbox[2] +
                ↪  random.uniform(0, max_r_trans)))
195             crop_ymax = max(h, int(max_bbox[3] +
                ↪  random.uniform(0, max_d_trans)))
196
197             image = image[crop_ymin : crop_ymax, crop_xmin
                ↪  : crop_xmax]
198
199             bboxes[:, [0, 2]] = bboxes[:, [0, 2]] -
                ↪  crop_xmin
200             bboxes[:, [1, 3]] = bboxes[:, [1, 3]] -
                ↪  crop_ymin
201
202         return image, bboxes
203
204     def random_translate(self, image, bboxes):
205         if random.random() < 0.5:
206             h, w, _ = image.shape
207             max_bbox = np.concatenate([np.min(bboxes[:,
                ↪  0:2], axis=0), np.max(bboxes[:, 2:4],
                ↪  axis=0)], axis=-1)
208
209             max_l_trans = max_bbox[0]
210             max_u_trans = max_bbox[1]
211             max_r_trans = w - max_bbox[2]
212             max_d_trans = h - max_bbox[3]
213
214             tx = random.uniform(-(max_l_trans - 1),
                ↪  (max_r_trans - 1))
215             ty = random.uniform(-(max_u_trans - 1),
                ↪  (max_d_trans - 1))
216
217             M = np.array([[1, 0, tx], [0, 1, ty]])
218             image = cv2.warpAffine(image, M, (w, h))
219
220             bboxes[:, [0, 2]] = bboxes[:, [0, 2]] + tx
221             bboxes[:, [1, 3]] = bboxes[:, [1, 3]] + ty
222
223         return image, bboxes
```

```
224
225     def parse_annotation(self, annotation, mAP = 'False'):
226         if TRAIN_LOAD_IMAGES_TO_RAM:
227             image_path = annotation[0]
228             image = annotation[2]
229         else:
230             image_path = annotation[0]
231             image = cv2.imread(image_path)
232
233         bboxes = np.array([list(map(int, box.split(',')))
             ↪ for box in annotation[1]])
234
235         if self.data_aug:
236             image, bboxes = self.random_horizontal_flip(np
                 ↪ .copy(image),
                 ↪ np.copy(bboxes))
237             image, bboxes =
                 ↪ self.random_crop(np.copy(image),
                 ↪ np.copy(bboxes))
238             image, bboxes =
                 ↪ self.random_translate(np.copy(image),
                 ↪ np.copy(bboxes))
239
240         #image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
241         if mAP == True:
242             return image, bboxes
243
244         image, bboxes = image_preprocess(np.copy(image),
             ↪ [self.input_sizes, self.input_sizes],
             ↪ np.copy(bboxes))
245         return image, bboxes
246
247     def preprocess_true_boxes(self, bboxes):
248         OUTPUT_LEVELS = len(self.strides)
249
250         label = [np.zeros((self.train_output_sizes[i],
             ↪ self.train_output_sizes[i],
             ↪ self.anchor_per_scale,
251                            5 + self.num_classes)) for i in
                             ↪ range(OUTPUT_LEVELS)]
```

```
252         bboxes_xywh = [np.zeros((self.max_bbox_per_scale,
             ↪ 4)) for _ in range(OUTPUT_LEVELS)]
253         bbox_count = np.zeros((OUTPUT_LEVELS,))
254
255         for bbox in bboxes:
256             bbox_coor = bbox[:4]
257             bbox_class_ind = bbox[4]
258
259             onehot = np.zeros(self.num_classes,
                 ↪ dtype=np.float)
260             onehot[bbox_class_ind] = 1.0
261             uniform_distribution =
                 ↪ np.full(self.num_classes, 1.0 /
                 ↪ self.num_classes)
262             deta = 0.01
263             smooth_onehot = onehot * (1 - deta) + deta *
                 ↪ uniform_distribution
264
265             bbox_xywh = np.concatenate([(bbox_coor[2:] +
                 ↪ bbox_coor[:2]) * 0.5, bbox_coor[2:] -
                 ↪ bbox_coor[:2]], axis=-1)
266             bbox_xywh_scaled = 1.0 * bbox_xywh[np.newaxis,
                 ↪ :] / self.strides[:, np.newaxis]
267
268             iou = []
269             exist_positive = False
270             for i in range(OUTPUT_LEVELS):#range(3):
271                 anchors_xywh =
                     ↪ np.zeros((self.anchor_per_scale, 4))
272                 anchors_xywh[:, 0:2] =
                     ↪ np.floor(bbox_xywh_scaled[i,
                     ↪ 0:2]).astype(np.int32) + 0.5
273                 anchors_xywh[:, 2:4] = self.anchors[i]
274
275                 iou_scale = bbox_iou(bbox_xywh_scaled[i][n
                     ↪ p.newaxis, :],
                     ↪ anchors_xywh)
276                 iou.append(iou_scale)
277                 iou_mask = iou_scale > 0.3
278
279                 if np.any(iou_mask):
```

```
                    xind, yind =
                     ↪  np.floor(bbox_xywh_scaled[i,
                     ↪  0:2]).astype(np.int32)

                    label[i][yind, xind, iou_mask, :] = 0
                    label[i][yind, xind, iou_mask, 0:4] =
                     ↪  bbox_xywh
                    label[i][yind, xind, iou_mask, 4:5] =
                     ↪  1.0
                    label[i][yind, xind, iou_mask, 5:] =
                     ↪  smooth_onehot

                    bbox_ind = int(bbox_count[i] %
                     ↪  self.max_bbox_per_scale)
                    bboxes_xywh[i][bbox_ind, :4] =
                     ↪  bbox_xywh
                    bbox_count[i] += 1

                    exist_positive = True

            if not exist_positive:
                best_anchor_ind =
                 ↪  np.argmax(np.array(iou).reshape(-1),
                 ↪  axis=-1)
                best_detect = int(best_anchor_ind /
                 ↪  self.anchor_per_scale)
                best_anchor = int(best_anchor_ind %
                 ↪  self.anchor_per_scale)
                xind, yind =
                 ↪  np.floor(bbox_xywh_scaled[best_detect,
                 ↪  0:2]).astype(np.int32)

                label[best_detect][yind, xind,
                 ↪  best_anchor, :] = 0
                label[best_detect][yind, xind,
                 ↪  best_anchor, 0:4] = bbox_xywh
                label[best_detect][yind, xind,
                 ↪  best_anchor, 4:5] = 1.0
                label[best_detect][yind, xind,
                 ↪  best_anchor, 5:] = smooth_onehot

```

```
304                 bbox_ind = int(bbox_count[best_detect] %
                     ↪ self.max_bbox_per_scale)
305                 bboxes_xywh[best_detect][bbox_ind, :4] =
                     ↪ bbox_xywh
306                 bbox_count[best_detect] += 1
307
308         if self.train_yolo_tiny:
309             label_mbbox, label_lbbox = label
310             mbboxes, lbboxes = bboxes_xywh
311             return label_mbbox, label_lbbox, mbboxes,
                 ↪ lbboxes
312
313         label_sbbox, label_mbbox, label_lbbox = label
314         sbboxes, mbboxes, lbboxes = bboxes_xywh
315         return label_sbbox, label_mbbox, label_lbbox,
             ↪ sbboxes, mbboxes, lbboxes
316
317     def __len__(self):
318         return self.num_batchs
```

C.3.5 train.py

```
#================================================================
#
#   File name   : train.py
#   Author      : PyLessons
#   Created date: 2020-08-06
#   Website     : https://pylessons.com/
#   GitHub      : https://github.com/pythonlessons/TensorFlow-2.x-YOLOv3
#   Description : used to train custom object detector
#        Modified by : Virginia Ramón
#================================================================

# Coments added by Virginia Ramon
import os
os.environ['CUDA_VISIBLE_DEVICES'] = '0,1'
os.environ['TF_FORCE_GPU_ALLOW_GROWTH'] = 'true'
from tensorflow.python.client import device_lib
import shutil
import numpy as np
import tensorflow as tf
config = tf.compat.v1.ConfigProto()
config.gpu_options.allow_growth = True
session = tf.compat.v1.Session(config=config)
from yolov3.dataset import Dataset
from yolov3.yolov4 import Create_Yolo, compute_loss
from yolov3.utils import load_yolo_weights
from yolov3.configs import *
from evaluate_mAP import get_mAP


# Define YOLO architecture
if YOLO_TYPE == "yolov4":
    print("YOLOv4...")
    Darknet_weights = YOLO_V4_TINY_WEIGHTS if TRAIN_YOLO_TINY else YOLO_V4_WEIGHTS
if YOLO_TYPE == "yolov3":
    print("YOLOv3...")
```

```
36      Darknet_weights = YOLO_V3_TINY_WEIGHTS if
        ↪ TRAIN_YOLO_TINY else YOLO_V3_WEIGHTS
37  if TRAIN_YOLO_TINY: TRAIN_MODEL_NAME += "_Tiny"
38
39  print("Num GPUs Available: ",
    ↪ len(tf.config.list_physical_devices('GPU')))
40  def main():
41          # Create register file for training process
            ↪ (Virginia Ramón)
42      file = open("loss_training_PigPerse46.txt", 'w')
43      print("Register file created!")
44
45      global TRAIN_FROM_CHECKPOINT
46
47      #Allow memmory growth in GPUs
48      gpus =
        ↪ tf.config.experimental.list_physical_devices('GPU')
49      for gpu in gpus:
50        tf.config.experimental.set_memory_growth(gpu, True)
51
52      # Save training summary
53      if os.path.exists(TRAIN_LOGDIR):
        ↪ shutil.rmtree(TRAIN_LOGDIR)
54      writer = tf.summary.create_file_writer(TRAIN_LOGDIR)
55
56      # Import train and validation datasets
57      print("Reading train dataset...")
58      trainset = Dataset('train')
59      print("Reading validation dataset...")
60      testset = Dataset('test')
61
62      # Define epochs and lengths
63      steps_per_epoch = len(trainset)
64      global_steps = tf.Variable(1, trainable=False,
        ↪ dtype=tf.int64)
65      warmup_steps = TRAIN_WARMUP_EPOCHS * steps_per_epoch
66      total_steps = TRAIN_EPOCHS * steps_per_epoch
67
68      # Recreate YOLOv4 with Darknet weights
69      if TRAIN_TRANSFER:
70          print("Creating Darknet model...")
```

```
71          Darknet = Create_Yolo(input_size=YOLO_INPUT_SIZE,
             ↪  CLASSES=YOLO_COCO_CLASSES)
72          print("Loading model weights Darknet...")
73          load_yolo_weights(Darknet, Darknet_weights) # use
             ↪  darknet weights
74
75      # Create YOLOv4 to train
76      print("Creating YOLO model...")
77      yolo = Create_Yolo(input_size=YOLO_INPUT_SIZE,
         ↪  training=True, CLASSES=TRAIN_CLASSES)
78      if TRAIN_FROM_CHECKPOINT:
79          try:
80              print("Loading checkpoint weights...")
81              yolo.load_weights(f"./checkpoints/yolov4_PigPe
                 ↪  rsv2/{TRAIN_MODEL_NAME}")
82          except ValueError:
83              print("Shapes are incompatible, transfering
                 ↪  Darknet weights")
84              TRAIN_FROM_CHECKPOINT = False
85
86      if TRAIN_TRANSFER and not TRAIN_FROM_CHECKPOINT:
87          print("Loading darknet weights to model...")
88          for i, l in enumerate(Darknet.layers):
89              layer_weights = l.get_weights()
90              if layer_weights != []:
91                  try:
92                      yolo.layers[i].set_weights(layer_weigh
                         ↪  ts)
93                  except:
94                      print("skipping", yolo.layers[i].name)
95
96      # Choose the optimizer for the training
97      optimizer = tf.keras.optimizers.Adam()
98
99      # Training step: for each image in train dataset
         ↪  compute gradient and apply to NN with LR
100     def train_step(image_data, target):
101         with tf.GradientTape() as tape:
102             pred_result = yolo(image_data, training=True)
103             giou_loss=conf_loss=prob_loss=0
104
```

```
105             # optimizing process
106             grid = 3 if not TRAIN_YOLO_TINY else 2
107             for i in range(grid):
108                 conv, pred = pred_result[i*2],
                     ↪ pred_result[i*2+1]
109                 loss_items = compute_loss(pred, conv,
                     ↪ *target[i], i, CLASSES=TRAIN_CLASSES)
110                 giou_loss += loss_items[0]
111                 conf_loss += loss_items[1]
112                 prob_loss += loss_items[2]
113
114             total_loss = giou_loss + conf_loss + prob_loss
115
116             gradients = tape.gradient(total_loss,
                 ↪ yolo.trainable_variables)
117             optimizer.apply_gradients(zip(gradients,
                 ↪ yolo.trainable_variables))
118
119             # update learning rate
120             # about warmup:
                 ↪ https://arxiv.org/pdf/1812.01187.pdf&usg=A
                 ↪ LkJrhglKOPDjNt6SHGbphTHyMcT0cuMJg
121             global_steps.assign_add(1)
122             if global_steps < warmup_steps:# and not
                 ↪ TRAIN_TRANSFER:
123                 lr = global_steps / warmup_steps *
                     ↪ TRAIN_LR_INIT
124             else:
125                 lr = TRAIN_LR_END + 0.5 * (TRAIN_LR_INIT -
                     ↪ TRAIN_LR_END)*(
126                     (1 + tf.cos((global_steps -
                         ↪ warmup_steps) / (total_steps -
                         ↪ warmup_steps) * np.pi)))
127             optimizer.lr.assign(lr.numpy())
128
129             # writing summary data
130             with writer.as_default():
131                 tf.summary.scalar("lr", optimizer.lr,
                     ↪ step=global_steps)
132                 tf.summary.scalar("loss/total_loss",
                     ↪ total_loss, step=global_steps)
```

```
133                 tf.summary.scalar("loss/giou_loss",
                     ↪  giou_loss, step=global_steps)
134                 tf.summary.scalar("loss/conf_loss",
                     ↪  conf_loss, step=global_steps)
135                 tf.summary.scalar("loss/prob_loss",
                     ↪  prob_loss, step=global_steps)
136             writer.flush()
137
138         return global_steps.numpy(), optimizer.lr.numpy(),
             ↪  giou_loss.numpy(), conf_loss.numpy(),
             ↪  prob_loss.numpy(), total_loss.numpy()
139
140     validate_writer =
         ↪  tf.summary.create_file_writer(TRAIN_LOGDIR)
141     def validate_step(image_data, target):
142         with tf.GradientTape() as tape:
143             pred_result = yolo(image_data, training=False)
144             giou_loss=conf_loss=prob_loss=0
145
146             # optimizing process
147             grid = 3 if not TRAIN_YOLO_TINY else 2
148             for i in range(grid):
149                 conv, pred = pred_result[i*2],
                     ↪  pred_result[i*2+1]
150                 loss_items = compute_loss(pred, conv,
                     ↪  *target[i], i, CLASSES=TRAIN_CLASSES)
151                 giou_loss += loss_items[0]
152                 conf_loss += loss_items[1]
153                 prob_loss += loss_items[2]
154
155             total_loss = giou_loss + conf_loss + prob_loss
156
157         return giou_loss.numpy(), conf_loss.numpy(),
             ↪  prob_loss.numpy(), total_loss.numpy()
158
159     mAP_model = Create_Yolo(input_size=YOLO_INPUT_SIZE,
         ↪  CLASSES=TRAIN_CLASSES) # create second model to
         ↪  measure mAP
160
161     print("Start training!")
162     with tf.device('/device:GPU:1'):
```

```
163         best_val_loss = 1000 # should be large at start
164         for epoch in range(TRAIN_EPOCHS):
165             for image_data, target in trainset:
166                 results = train_step(image_data, target)
167                 cur_step = results[0]%steps_per_epoch
168                 print("epoch:{:2.0f} step:{:5.0f}/{},
                     ↪ lr:{:.6f}, giou_loss:{:7.2f},
                     ↪ conf_loss:{:7.2f}, prob_loss:{:7.2f},
                     ↪ total_loss:{:7.2f}"
169                       .format(epoch, cur_step,
                           ↪ steps_per_epoch, results[1],
                           ↪ results[2], results[3],
                           ↪ results[4], results[5]))
170
171             if len(testset) == 0:
172                 print("configure TEST options to validate
                     ↪ model")
173                 yolo.save_weights(os.path.join(TRAIN_CHECK
                     ↪ POINTS_FOLDER,
                     ↪ TRAIN_MODEL_NAME))
174                 continue
175
176             print("Validating results...")
177             count, giou_val, conf_val, prob_val, total_val
                 ↪ = 0., 0, 0, 0, 0
178             for image_data, target in testset:
179                 results = validate_step(image_data, target)
180                 count += 1
181                 giou_val += results[0]
182                 conf_val += results[1]
183                 prob_val += results[2]
184                 total_val += results[3]
185             # writing validate summary data
186             with validate_writer.as_default():
187                 tf.summary.scalar("validate_loss/total_val
                     ↪ ", total_val/count,
                     ↪ step=epoch)
188                 tf.summary.scalar("validate_loss/giou_val"
                     ↪ , giou_val/count,
                     ↪ step=epoch)
```

```
189                 tf.summary.scalar("validate_loss/conf_val"
                     ↪  , conf_val/count,
                     ↪  step=epoch)
190                 tf.summary.scalar("validate_loss/prob_val"
                     ↪  , prob_val/count,
                     ↪  step=epoch)
191             validate_writer.flush()
192             file.write(str(epoch)+","+str(giou_val/count)+
                 ↪  ","+str(conf_val/count)+","+str(prob_val/c
                 ↪  ount)+","+str(total_val/count)+"\n")
193             file.flush()
194
195             print("\n\ngiou_val_loss:{:7.2f},
                 ↪  conf_val_loss:{:7.2f},
                 ↪  prob_val_loss:{:7.2f},
                 ↪  total_val_loss:{:7.2f}\n\n".
196                   format(giou_val/count, conf_val/count,
                       ↪  prob_val/count, total_val/count))
197
198             if TRAIN_SAVE_CHECKPOINT and not
                 ↪  TRAIN_SAVE_BEST_ONLY:
199                 save_directory =
                     ↪  os.path.join(TRAIN_CHECKPOINTS_FOLDER,
                     ↪  TRAIN_MODEL_NAME+"_val_loss_{:7.2f}".f
                     ↪  ormat(total_val/count))
200                 yolo.save_weights(save_directory)
201                 print("Weights saved checkpoint!")
202             if TRAIN_SAVE_BEST_ONLY and
                 ↪  best_val_loss>total_val/count:
203                 save_directory =
                     ↪  os.path.join(TRAIN_CHECKPOINTS_FOLDER,
                     ↪  TRAIN_MODEL_NAME)
204                 yolo.save_weights(save_directory)
205                 best_val_loss = total_val/count
206                 print("Weights saved best checkpoint!")
207             if not TRAIN_SAVE_BEST_ONLY and not
                 ↪  TRAIN_SAVE_CHECKPOINT:
208                 save_directory =
                     ↪  os.path.join(TRAIN_CHECKPOINTS_FOLDER,
                     ↪  TRAIN_MODEL_NAME)
209                 yolo.save_weights(save_directory)
```

```
210                 print("Weights saved!")
211     file.close()
212     # measure mAP of trained custom model
213     try:
214         mAP_model.load_weights(save_directory) # use keras
            ↪ weights
215         get_mAP(mAP_model, testset,
            ↪ score_threshold=TEST_SCORE_THRESHOLD,
            ↪ iou_threshold=TEST_IOU_THRESHOLD)
216     except UnboundLocalError:
217         print("You don't have saved model weights to
            ↪ measure mAP, check TRAIN_SAVE_BEST_ONLY and
            ↪ TRAIN_SAVE_CHECKPOINT lines in configs.py")
218
219 if __name__ == '__main__':
220     main()
```

C.3.6 detection_demo.py

```
#=============================================================
#
#   File name   : detection_demo.py
#   Author      : PyLessons
#   Created date: 2020-09-27
#   Website     : https://pylessons.com/
#   GitHub      : https://github.com/pythonlessons/TensorFlow-2.x-YOLOv3
#   Description : object detection image and video example
#   Modified by : Virginia Ramón
#=============================================================
import os
os.environ['CUDA_VISIBLE_DEVICES'] = '0'
import cv2
import numpy as np
import tensorflow as tf
physical_devices = tf.config.list_physical_devices('GPU')
for gpu_instance in physical_devices:
    tf.config.experimental.set_memory_growth(gpu_instance, True)
from yolov3.utils import detect_image, detect_realtime, detect_video, Load_Yolo_model, detect_video_realtime_mp
from yolov3.configs import *

# Define if want to detect image or video
img = False

# Load model
yolo = Load_Yolo_model()

# Image detection
if img:
    image_path   = "/home/virginia/TFG/PIGDATA2/frames/GH010013_zenital_Val_2220.jpg"
    image_out_path = "./IMAGES/GH010013_zenital_Val_pred_05.jpg"
```

```
32      _, data, classes = detect_image(yolo, image_path,
        ↪  image_out_path, input_size=YOLO_INPUT_SIZE,
        ↪  show=False, score_threshold=0.5,
        ↪  rectangle_colors=(255,0,0))
33
34      # Data to txt
35      datafile = []
36      fr = str(data[0])
37      bbxs = ""
38      for bb in data[1]:
39          pos = str(bb[0])+","+str(bb[1])+","+str(bb[2])+","
            ↪  +str(bb[3])+","+str(bb[5])
40          bbxs= bbxs + " " + pos
41      datafile = fr+" "+bbxs+"\n"
42
43      file = open("./IMAGES/pred_img.txt", 'w')
44      file.write(datafile)
45      file.close()
46      print("Image detection ended")
47  # Video detection
48  else:
49      video_path   = "/home/virginia/TFG/PIGDATA/annotated/2
        ↪  019_12_10/000078/color_mask.mp4"
50      video_out_path = "./IMAGES/00078_color_mask_pred.mp4"
51      data, classes = detect_video(yolo, video_path,
        ↪  video_out_path, input_size=YOLO_INPUT_SIZE,
        ↪  show=False, score_threshold=0.5,
        ↪  rectangle_colors=(255,0,0))
52
53      # Data to txt
54      datafile = []
55      for frame in data:
56          fr = str(frame[1])
57          bbxs = ""
58          for bb in frame[2]:
59              pos = str(bb[0])+","+str(bb[1])+","+str(bb[2])
                ↪  +","+str(bb[3])+","+str(bb[5])
60              bbxs= bbxs + " " + pos
61          datafile.append(fr+" "+bbxs)
62
63      file = open("./IMAGES/pred_video.txt", 'w')
```

```
64      for d in datafile:
65          file.write(d+"\n")
66      file.close()
67      print("Video detection ended")
```

C.3.7 evaluate_mAP.py

```
#================================================================
#
#   File name   : evaluate_mAP.py
#   Author      : PyLessons
#   Created date: 2020-08-17
#   Website     : https://pylessons.com/
#   GitHub      : https://github.com/pythonlessons/TensorFlow-2.x-YOLOv3
#   Description : used to evaluate model mAP and FPS
#
#================================================================
import os
os.environ['CUDA_VISIBLE_DEVICES'] = '0'
import cv2
import numpy as np
import tensorflow as tf
from tensorflow.python.saved_model import tag_constants
from yolov3.dataset import Dataset
from yolov3.yolov4 import Create_Yolo
from yolov3.utils import load_yolo_weights, detect_image, image_preprocess, postprocess_boxes, nms, read_class_names
from yolov3.configs import *
import shutil
import json
import time

gpus = tf.config.experimental.list_physical_devices('GPU')
if len(gpus) > 0:
    try: tf.config.experimental.set_memory_growth(gpus[0], True)
    except RuntimeError: print("RuntimeError in tf.config.experimental.list_physical_devices('GPU')")


def voc_ap(rec, prec):
    """
```

```
    --- Official matlab code VOC2012---
    mrec=[0 ; rec ; 1];
    mpre=[0 ; prec ; 0];
    for i=numel(mpre)-1:-1:1
            mpre(i)=max(mpre(i),mpre(i+1));
    end
    i=find(mrec(2:end)~=mrec(1:end-1))+1;
    ap=sum((mrec(i)-mrec(i-1)).*mpre(i));
    """
    rec.insert(0, 0.0) # insert 0.0 at begining of list
    rec.append(1.0) # insert 1.0 at end of list
    mrec = rec[:]
    prec.insert(0, 0.0) # insert 0.0 at begining of list
    prec.append(0.0) # insert 0.0 at end of list
    mpre = prec[:]
    """
     This part makes the precision monotonically decreasing
        (goes from the end to the beginning)
        matlab:  for i=numel(mpre)-1:-1:1

    mpre(i)=max(mpre(i),mpre(i+1));
    """
    # matlab indexes start in 1 but python in 0, so I have
     to do:
    #   range(start=(len(mpre) - 2), end=0, step=-1)
    # also the python function range excludes the end,
     resulting in:
    #   range(start=(len(mpre) - 2), end=-1, step=-1)
    for i in range(len(mpre)-2, -1, -1):
        mpre[i] = max(mpre[i], mpre[i+1])
    """
     This part creates a list of indexes where the recall
    changes
        matlab:  i=find(mrec(2:end)~=mrec(1:end-1))+1;
    """
    i_list = []
    for i in range(1, len(mrec)):
        if mrec[i] != mrec[i-1]:
            i_list.append(i) # if it was matlab would be i
             + 1
    """
```

```
69          The Average Precision (AP) is the area under the curve
70              (numerical integration)
71              matlab: ap=sum((mrec(i)-mrec(i-1)).*mpre(i));
72          """
73          ap = 0.0
74          for i in i_list:
75              ap += ((mrec[i]-mrec[i-1])*mpre[i])
76          return ap, mrec, mpre
77
78
79      def get_mAP(Yolo, dataset, score_threshold=0.4,
        ↪  iou_threshold=0.4, TEST_INPUT_SIZE=TEST_INPUT_SIZE):
80          MINOVERLAP = 0.5 # default value (defined in the
            ↪  PASCAL VOC2012 challenge)
81          NUM_CLASS = read_class_names(TRAIN_CLASSES)
82
83          ground_truth_dir_path = 'mAP/ground-truth'
84          if os.path.exists(ground_truth_dir_path):
            ↪  shutil.rmtree(ground_truth_dir_path)
85
86          if not os.path.exists('mAP'): os.mkdir('mAP')
87          os.mkdir(ground_truth_dir_path)
88
89          print(f'\ncalculating
            ↪  mAP{int(iou_threshold*100)}...\n')
90
91          gt_counter_per_class = {}
92          for index in range(dataset.num_samples):
93              print(index)
94              ann_dataset = dataset.annotations[index]
95
96              original_image, bbox_data_gt =
                ↪  dataset.parse_annotation(ann_dataset, True)
97
98              if len(bbox_data_gt) == 0:
99                  bboxes_gt = []
100                 classes_gt = []
101             else:
102                 bboxes_gt, classes_gt = bbox_data_gt[:, :4],
                    ↪  bbox_data_gt[:, 4]
```

```
103         ground_truth_path =
             ↪ os.path.join(ground_truth_dir_path, str(index)
             ↪ + '.txt')
104         num_bbox_gt = len(bboxes_gt)
105
106         bounding_boxes = []
107         for i in range(num_bbox_gt):
108             class_name = NUM_CLASS[classes_gt[i]]
109             xmin, ymin, xmax, ymax = list(map(str,
                 ↪ bboxes_gt[i]))
110             bbox = xmin + " " + ymin + " " + xmax + " "
                 ↪ +ymax
111             bounding_boxes.append({"class_name":class_name
                 ↪ , "bbox":bbox,
                 ↪ "used":False})
112
113             # count that object
114             if class_name in gt_counter_per_class:
115                 gt_counter_per_class[class_name] += 1
116             else:
117                 # if class didn't exist yet
118                 gt_counter_per_class[class_name] = 1
119             bbox_mess = ' '.join([class_name, xmin, ymin,
                 ↪ xmax, ymax]) + '\n'
120         with open(f'{ground_truth_dir_path}/{str(index)}_g
             ↪ round_truth.json', 'w') as
             ↪ outfile:
121             json.dump(bounding_boxes, outfile)
122
123     gt_classes = list(gt_counter_per_class.keys())
124     # sort the classes alphabetically
125     gt_classes = sorted(gt_classes)
126     n_classes = len(gt_classes)
127
128     times = []
129     json_pred = [[] for i in range(n_classes)]
130     for index in range(dataset.num_samples):
131         ann_dataset = dataset.annotations[index]
132
133         image_name = ann_dataset[0].split('/')[-1]
```

```
134         original_image, bbox_data_gt =
             ↪  dataset.parse_annotation(ann_dataset, True)
135
136         image = image_preprocess(np.copy(original_image),
             ↪  [TEST_INPUT_SIZE, TEST_INPUT_SIZE])
137         image_data = image[np.newaxis,
             ↪  ...].astype(np.float32)
138
139         t1 = time.time()
140         if YOLO_FRAMEWORK == "tf":
141             pred_bbox = Yolo.predict(image_data)
142         elif YOLO_FRAMEWORK == "trt":
143             batched_input = tf.constant(image_data)
144             result = Yolo(batched_input)
145             pred_bbox = []
146             for key, value in result.items():
147                 value = value.numpy()
148                 pred_bbox.append(value)
149
150         t2 = time.time()
151
152         times.append(t2-t1)
153
154         pred_bbox = [tf.reshape(x, (-1, tf.shape(x)[-1]))
             ↪  for x in pred_bbox]
155         pred_bbox = tf.concat(pred_bbox, axis=0)
156
157         bboxes = postprocess_boxes(pred_bbox,
             ↪  original_image, TEST_INPUT_SIZE,
             ↪  score_threshold)
158         bboxes = nms(bboxes, iou_threshold, method='nms')
159
160         for bbox in bboxes:
161             coor = np.array(bbox[:4], dtype=np.int32)
162             score = bbox[4]
163             class_ind = int(bbox[5])
164             class_name = NUM_CLASS[class_ind]
165             score = '%.4f' % score
166             xmin, ymin, xmax, ymax = list(map(str, coor))
167             bbox = xmin + " " + ymin + " " + xmax + " "
                 ↪  +ymax
```

```
168             json_pred[gt_classes.index(class_name)].append
                ↪  ({"confidence": str(score), "file_id":
                ↪  str(index), "bbox": str(bbox)})
169
170     ms = sum(times)/len(times)*1000
171     fps = 1000 / ms
172
173     for class_name in gt_classes:
174         json_pred[gt_classes.index(class_name)].sort(key=l
            ↪  ambda x:float(x['confidence']),
            ↪  reverse=True)
175         with open(f'{ground_truth_dir_path}/{class_name}_p
            ↪  redictions.json', 'w') as
            ↪  outfile:
176             json.dump(json_pred[gt_classes.index(class_nam
                ↪  e)],
                ↪  outfile)
177
178     # Calculate the AP for each class
179     sum_AP = 0.0
180     ap_dictionary = {}
181     # open file to store the results
182     with open("mAP/results.txt", 'w') as results_file:
183         results_file.write("# AP and precision/recall per
            ↪  class\n")
184         count_true_positives = {}
185         for class_index, class_name in
            ↪  enumerate(gt_classes):
186             count_true_positives[class_name] = 0
187             # Load predictions of that class
188             predictions_file = f'{ground_truth_dir_path}/{
                ↪  class_name}_predictions.json'
189             predictions_data =
                ↪  json.load(open(predictions_file))
190
191             # Assign predictions to ground truth objects
192             nd = len(predictions_data)
193             tp = [0] * nd # creates an array of zeros of
                ↪  size nd
194             fp = [0] * nd
195             tp_count = 0
```

```
196             fp_count = 0
197             for idx, prediction in
                ↪  enumerate(predictions_data):
198                 file_id = prediction["file_id"]
199                 # assign prediction to ground truth object
                    ↪  if any
200                 #   open ground-truth with that file_id
201                 gt_file = f'{ground_truth_dir_path}/{str(f
                    ↪  ile_id)}_ground_truth.json'
202                 ground_truth_data =
                    ↪  json.load(open(gt_file))
203                 ovmax = -1
204                 gt_match = -1
205                 # load prediction bounding-box
206                 bb = [ float(x) for x in
                    ↪  prediction["bbox"].split() ] #
                    ↪  bounding box of prediction
207                 for obj in ground_truth_data:
208                     # look for a class_name match
209                     if obj["class_name"] == class_name:
210                         bbgt = [ float(x) for x in
                            ↪  obj["bbox"].split() ] #
                            ↪  bounding box of ground truth
211                         bi = [max(bb[0],bbgt[0]),
                            ↪  max(bb[1],bbgt[1]),
                            ↪  min(bb[2],bbgt[2]),
                            ↪  min(bb[3],bbgt[3])]
212                         iw = bi[2] - bi[0] + 1
213                         ih = bi[3] - bi[1] + 1
214                         if iw > 0 and ih > 0:
215                             # compute overlap (IoU) = area
                                ↪  of intersection / area of
                                ↪  union
216                             ua = (bb[2] - bb[0] + 1) *
                                ↪  (bb[3] - bb[1] + 1) +
                                ↪  (bbgt[2] - bbgt[0]
217                                                 + 1) *
                                    ↪  (bbgt[3] -
                                    ↪  bbgt[1] +
                                    ↪  1) - iw *
                                    ↪  ih
```

```
                        ov = iw * ih / ua
                        if ov > ovmax:
                            ovmax = ov
                            gt_match = obj

            # assign prediction as true positive/don't care/false positive
            if ovmax >= MINOVERLAP:# if ovmax > minimum overlap
                if not bool(gt_match["used"]):
                    # true positive
                    tp[idx] = 1
                    tp_count += 1
                    gt_match["used"] = True
                    count_true_positives[class_name] += 1
                    # update the ".json" file
                    with open(gt_file, 'w') as f:
                        f.write(json.dumps(ground_truth_data))
                else:
                    # false positive (multiple detection)
                    fp[idx] = 1
                    fp_count += 1
            else:
                # false positive
                fp[idx] = 1
                fp_count += 1

        # compute precision/recall
        cumsum = 0

        for idx, val in enumerate(fp):
            fp[idx] += cumsum
            cumsum += val
        cumsum = 0
        for idx, val in enumerate(tp):
            tp[idx] += cumsum
            cumsum += val
        #print(tp)
```

```
254             rec = tp[:]
255             for idx, val in enumerate(tp):
256                 rec[idx] = float(tp[idx]) /
                     ↪ gt_counter_per_class[class_name]
257             #print(rec)
258             prec = tp[:]
259             for idx, val in enumerate(tp):
260                 prec[idx] = float(tp[idx]) / (fp[idx] +
                     ↪ tp[idx])
261             #print(prec)
262
263             ap, mrec, mprec = voc_ap(rec, prec)
264             sum_AP += ap
265             text = "{0:.3f}%".format(ap*100) + " = " +
                 ↪ class_name + " AP  " #class_name + " AP =
                 ↪ {0:.2f}%".format(ap*100)
266
267             rounded_prec = [ '%.3f' % elem for elem in prec
                 ↪ ]
268             rounded_rec = [ '%.3f' % elem for elem in rec ]
269             # Write to results.txt
270             results_file.write(text + "\n Precision: " +
                 ↪ str(rounded_prec) + "\n Recall   :" +
                 ↪ str(rounded_rec) + "\n\n")
271
272             print(text)
273             print("{0:.1f}".format(tp_count) + " = " +
                 ↪ class_name + " TP  ")
274             print("{0:.1f}".format(fp_count) + " = " +
                 ↪ class_name + " FP  ")
275             ap_dictionary[class_name] = ap
276
277         results_file.write("\n# mAP of all classes\n")
278         mAP = sum_AP / n_classes
279
280         text = "mAP = {:.3f}%, {:.2f} FPS".format(mAP*100,
             ↪ fps)
281         results_file.write(text + "\n")
282         print(text)
283
284         return mAP*100
```

```
285
286 if __name__ == '__main__':
287     if YOLO_FRAMEWORK == "tf": # TensorFlow detection
288         if YOLO_TYPE == "yolov4":
289             Darknet_weights = YOLO_V4_TINY_WEIGHTS if
                ↪ TRAIN_YOLO_TINY else YOLO_V4_WEIGHTS
290         if YOLO_TYPE == "yolov3":
291             Darknet_weights = YOLO_V3_TINY_WEIGHTS if
                ↪ TRAIN_YOLO_TINY else YOLO_V3_WEIGHTS
292
293         if YOLO_CUSTOM_WEIGHTS == False:
294             yolo = Create_Yolo(input_size=YOLO_INPUT_SIZE,
                ↪ CLASSES=YOLO_COCO_CLASSES)
295             load_yolo_weights(yolo, Darknet_weights) # use
                ↪ Darknet weights
296         else:
297             yolo = Create_Yolo(input_size=YOLO_INPUT_SIZE,
                ↪ CLASSES=TRAIN_CLASSES)
298             yolo.load_weights(f"./checkpoints/yolov4_PigPe
                ↪ rsv5/{TRAIN_MODEL_NAME}") # use custom
                ↪ weights
299
300     elif YOLO_FRAMEWORK == "trt": # TensorRT detection
301         saved_model_loaded = tf.saved_model.load(f"./check
            ↪ points/{TRAIN_MODEL_NAME}",
            ↪ tags=[tag_constants.SERVING])
302         signature_keys =
            ↪ list(saved_model_loaded.signatures.keys())
303         yolo = saved_model_loaded.signatures['serving_defa
            ↪ ult']
304
305     testset = Dataset('test',
        ↪ TEST_INPUT_SIZE=YOLO_INPUT_SIZE)
306     get_mAP(yolo, testset, score_threshold=0.4,
        ↪ iou_threshold=0.4, TEST_INPUT_SIZE=YOLO_INPUT_SIZE)
```

C.3.8 mAPtest.py

```
import os
os.environ['CUDA_VISIBLE_DEVICES'] = '0,1'
os.environ['TF_FORCE_GPU_ALLOW_GROWTH'] = 'true'
from tensorflow.python.client import device_lib
#print(device_lib.list_local_devices())
import shutil
import numpy as np
import tensorflow as tf
#gpu_options = tf.compat.v1.GPUOptions(per_process_gpu_mem
  ↪ ory_fraction=0.9)
#sess = tf.compat.v1.Session(config=tf.compat.v1.ConfigPro
  ↪ to(gpu_options=gpu_options))
#from tensorflow.keras.utils import plot_model
config = tf.compat.v1.ConfigProto()
config.gpu_options.allow_growth = True
session = tf.compat.v1.Session(config=config)
from yolov3.dataset import Dataset
from yolov3.yolov4 import Create_Yolo, compute_loss
from yolov3.utils import load_yolo_weights
from yolov3.configs import *
from evaluate_mAP import get_mAP

#Allow memmory growth in GPUs
gpus = tf.config.experimental.list_physical_devices('GPU')
for gpu in gpus:
  tf.config.experimental.set_memory_growth(gpu, True)

print("Reading test dataset...")
testset = Dataset('test')

save_directory = os.path.join(TRAIN_CHECKPOINTS_FOLDER,
  ↪ TRAIN_MODEL_NAME)

print("Creating model...")
mAP_model = Create_Yolo(input_size=YOLO_INPUT_SIZE,
  ↪ CLASSES=TRAIN_CLASSES) # create second model to
  ↪ measure mAP

try:
```

```
35          mAP_model.load_weights(save_directory) # use keras
             ↪ weights
36          print("Calculating mAP...")
37          get_mAP(mAP_model, testset,
             ↪ score_threshold=TEST_SCORE_THRESHOLD,
             ↪ iou_threshold=TEST_IOU_THRESHOLD)
38  except UnboundLocalError:
39          print("You don't have saved model weights to
             ↪ measure mAP, check TRAIN_SAVE_BEST_ONLY and
             ↪ TRAIN_SAVE_CHECKPOINT lines in configs.py")
40
```

D.4 Codi UKF

D.4.1 ukf.py

```
'''
    File name          : ukf.py
    Author             : Srini Ananthakrishnan
    Python Version     : 2.7
'''

import numpy as np
import scipy.linalg
from copy import deepcopy
from threading import Lock


class UKFException(Exception):
    """Raise for errors in the UKF, usually due to bad
     ↪  inputs"""


class UKF:
    def __init__(self, initial_state):#,iterate_function):
        """
        Initializes the unscented kalman filter
        :param num_states: int, the size of the state
        :param process_noise: the process noise covariance
 ↪  per unit time, should be num_states x num_states
        :param initial_state: initial values for the
 ↪  states, should be num_states x 1
        :param initial_covar: initial covariance matrix,
 ↪  should be num_states x num_states, typically large and
 ↪  diagonal
        :param alpha: UKF tuning parameter, determines
 ↪  spread of sigma points, typically a small positive
 ↪  value
        :param k: UKF tuning parameter, typically 0 or 3 -
 ↪  num_states
        :param beta: UKF tuning parameter, beta = 2 is
 ↪  ideal for gaussian distributions
```

```
28          :param iterate_function: function that predicts
    ↪ the next state
29                      takes in a num_states x 1 state and a
    ↪ float timestep
30                      returns a num_states x 1 state
31          """
32          num_states=6
33          self.n_dim = int(num_states)
34          #self.n_sig = 1 + num_states * 2
35          self.n_sig = 2 * num_states + 1
36          #self.q = process_noise
37          self.q = np.eye(6)
38                  # ---------------------- MODIFIED BY
                     ↪ VIRGINIA RAMÓN ------------------
39          self.q[0][0] = 0.01
40          self.q[1][1] = 0.01
41          self.q[2][2] = 0.01
42          self.q[3][3] = 0.01
43          self.q[4][4] = 0.01
44          self.q[5][5] = 0.01
45
46          #self.x = initial_state
47          self.x = np.array([initial_state[0],initial_state[
             ↪ 1],0.5,0.5,0.5,0.5])
48          self.p = 0.1*np.diag((1.0, 1.0, 1.0, 1.0, 1.0,
             ↪ 1.0))
49          self.beta = 2
50          self.alpha = 0.04
51          self.k = 0
52          #self.iterate = iterate_function
53          self.lastResult = np.array([[0],
             ↪ [255],[0],[0],[0],[0]])
54          self.lambd = pow(self.alpha, 2) * (self.n_dim +
             ↪ self.k) - self.n_dim
55
56          self.covar_weights = np.zeros(self.n_sig)
57          self.mean_weights = np.zeros(self.n_sig)
58
59          self.covar_weights[0] = (self.lambd / (self.n_dim
             ↪ + self.lambd)) + (1 - pow(self.alpha, 2) +
             ↪ self.beta)
```

```
60          self.mean_weights[0] = (self.lambd / (self.n_dim +
             ↪  self.lambd))
61
62          for i in range(1, self.n_sig):
63              self.covar_weights[i] = 1 / (2*(self.n_dim +
                 ↪  self.lambd))
64              self.mean_weights[i] = 1 / (2*(self.n_dim +
                 ↪  self.lambd))
65
66          self.sigmas = self.__get_sigmas()
67
68          self.lock = Lock()
69
70      def iterate_x(self,x_in, timestep):
71          '''this function is based on the x_dot and can be
             ↪  nonlinear as needed'''
72          ret = np.zeros(len(x_in))
73          ret[0] = x_in[0] + timestep * x_in[2] + timestep *
             ↪  timestep * x_in[4]
74          ret[1] = x_in[1] + timestep * x_in[3] + timestep *
             ↪  timestep * x_in[5]
75          ret[2] = x_in[2] + timestep * x_in[4]
76          ret[3] = x_in[3] + timestep * x_in[5]
77          ret[4] = x_in[4]
78          ret[5] = x_in[5]
79          return ret
80
81      def __get_sigmas(self):
82          """generates sigma points"""
83          ret = np.zeros((self.n_sig, self.n_dim))
84
85          tmp_mat = (self.n_dim + self.lambd)*self.p
86
87          # print spr_mat
88          spr_mat = scipy.linalg.sqrtm(tmp_mat)
89
90          ret[0] = self.x
91          for i in range(self.n_dim):
92              ret[i+1] = np.ravel(self.x) + spr_mat[i]
93              ret[i+1+self.n_dim] = np.ravel(self.x) -
                 ↪  spr_mat[i]
```

```
94
95          return ret.T
96
97      def update(self, states, data, flag, r_matrix):
98          """
99          performs a measurement update
100         :param states: list of indices (zero-indexed) of
    ↪ which states were measured, that is, which are being
    ↪ updated
101         :param data: list of the data corresponding to the
    ↪ values in states
102         :param r_matrix: error matrix for the data, again
    ↪ corresponding to the values in states
103         """
104         if not flag:
105             data = np.array([self.lastResult[states]])
106             #print('Not detected, data is: ', data)
107
108         self.lock.acquire()
109
110         num_states = len(states)
111
112         # create y, sigmas of just the states that are
            ↪ being updated
113         sigmas_split = np.split(self.sigmas, self.n_dim)
114         y = np.concatenate([sigmas_split[i] for i in
            ↪ states])
115
116         # create y_mean, the mean of just the states that
            ↪ are being updated
117         x_split = np.split(self.x, self.n_dim)
118         y_mean = np.concatenate([x_split[i] for i in
            ↪ states])
119
120         # differences in y from y mean
121         y_diff = deepcopy(y)
122         x_diff = deepcopy(self.sigmas)
123         for i in range(self.n_sig):
124             for j in range(num_states):
125                 y_diff[j][i] -= y_mean[j]
126             for j in range(self.n_dim):
```

```
127                 x_diff[j][i] -= self.x[j]
128
129         # covariance of measurement
130         p_yy = np.zeros((num_states, num_states))
131         for i, val in enumerate(np.array_split(y_diff,
            ↪ self.n_sig, 1)):
132             p_yy += self.covar_weights[i] * val.dot(val.T)
133
134         # add measurement noise
135         p_yy += r_matrix
136
137         # covariance of measurement with states
138         p_xy = np.zeros((self.n_dim, num_states))
139         for i, val in enumerate(zip(np.array_split(y_diff,
            ↪ self.n_sig, 1), np.array_split(x_diff,
            ↪ self.n_sig, 1))):
140             p_xy += self.covar_weights[i] *
                ↪ val[1].dot(val[0].T)
141
142         k = np.dot(p_xy, np.linalg.inv(p_yy))
143
144         y_actual = np.ravel(data)
145
146         self.x += np.dot(k, (y_actual - y_mean))
147         self.p -= np.dot(k, np.dot(p_yy, k.T))
148         self.sigmas = self.__get_sigmas()
149         self.lastResult = self.x
150         self.lock.release()
151
152     '''
153     def update(self, data, flag, r_matrix):
154         """
155         performs a measurement update
156         :param states: list of indices (zero-indexed) of
    ↪ which states were measured, that is, which are being
    ↪ updated
157         :param data: list of the data corresponding to the
    ↪ values in states
158         :param r_matrix: error matrix for the data, again
    ↪ corresponding to the values in states
159         """
```

```
160
161         self.lock.acquire()
162         if not flag:
163             data = np.array([self.lastResult])
164         #num_states = len(states)
165
166         # create y, sigmas of just the states that are
    ↪ being updated
167         sigmas_split = np.split(self.sigmas, self.n_dim)
168         #y = np.concatenate([sigmas_split[i] for i in
    ↪ states])
169         y = sigmas_split
170
171         # create y_mean, the mean of just the states that
    ↪ are being updated
172         x_split = np.split(self.x, self.n_dim)
173         y_mean = x_split
174
175         # differences in y from y mean
176         y_diff = deepcopy(y)
177         x_diff = deepcopy(self.sigmas)
178         for i in range(self.n_sig):
179             for j in range(self.n_dim):
180                 y_diff[j][i] -= y_mean[j]
181             for j in range(self.n_dim):
182                 x_diff[j][i] -= self.x[j]
183
184         # covariance of measurement
185         p_yy = np.zeros(self.n_dim, self.n_dim)
186         for i, val in enumerate(np.array_split(y_diff,
    ↪ self.n_sig, 1)):
187             p_yy += self.covar_weights[i] * val.dot(val.T)
188
189         # add measurement noise
190         p_yy += r_matrix
191
192         # covariance of measurement with states
193         p_xy = np.zeros((self.n_dim, self.n_dim))
194         for i, val in enumerate(zip(np.array_split(y_diff,
    ↪ self.n_sig, 1), np.array_split(x_diff, self.n_sig,
    ↪ 1))):
```

```
195             p_xy += self.covar_weights[i] *
    ↪ val[1].dot(val[0].T)
196
197         k = np.dot(p_xy, np.linalg.inv(p_yy))
198
199         y_actual = data
200
201         self.x += np.dot(k, (y_actual - y_mean))
202         self.p -= np.dot(k, np.dot(p_yy, k.T))
203         self.sigmas = self.__get_sigmas()
204         print('r_matrix : ',r_matrix)
205
206         self.lock.release()
207     '''
208     def predict(self, timestep):
209         """
210         performs a prediction step
211         :param timestep: float, amount of time since last
    ↪ prediction
212         """
213
214         self.lock.acquire()
215         #print("x- shape: ",self.x.shape)
216         sigmas_out = np.array([self.iterate_x(x,timestep)
             ↪ for x in self.sigmas.T]).T
217
218         x_out = np.zeros(self.n_dim)
219
220         # for each variable in X
221         for i in range(self.n_dim):
222             # the mean of that variable is the sum of
223             # the weighted values of that variable for
                 ↪ each iterated sigma point
224             x_out[i] = sum((self.mean_weights[j] *
                 ↪ sigmas_out[i][j] for j in
                 ↪ range(self.n_sig)))
225
226         p_out = np.zeros((self.n_dim, self.n_dim))
227         # for each sigma point
228         for i in range(self.n_sig):
229             # take the distance from the mean
```

```
230             # make it a covariance by multiplying by the
                 ↪ transpose
231             # weight it using the calculated weighting
                 ↪ factor
232             # and sum
233             diff = sigmas_out.T[i] - x_out
234             diff = np.atleast_2d(diff)
235             p_out += self.covar_weights[i] *
                 ↪ np.dot(diff.T, diff)
236 
237         # add process noise
238         #p_out += timestep * self.q
239         p_out += self.q
240 
241         self.sigmas = sigmas_out
242         self.x = x_out
243         self.p = p_out
244         self.lastResult = self.x
245         self.lock.release()
246 
247     def get_state(self, index=-1):
248         """
249         returns the current state (n_dim x 1), or a
    ↪ particular state variable (float)
250         :param index: optional, if provided, the index of
    ↪ the returned variable
251         :return:
252         """
253         if index >= 0:
254             return self.x[index]
255         else:
256             return self.x
257 
258     def get_covar(self):
259         """
260         :return: current state covariance (n_dim x n_dim)
261         """
262         return self.p
263 
264     def set_state(self, value, index=-1):
265         """
```

```
266         Overrides the filter by setting one variable of
    ↪ the state or the whole state
267         :param value: the value to put into the state (1 x
    ↪ 1 or n_dim x 1)
268         :param index: the index at which to override the
    ↪ state (-1 for whole state)
269         """
270         with self.lock:
271             if index != -1:
272                 self.x[index] = value
273             else:
274                 self.x = value
275 
276     def reset(self, state, covar):
277         """
278         Restarts the UKF at the given state and covariance
279         :param state: n_dim x 1
280         :param covar: n_dim x n_dim
281         """
282 
283         with self.lock:
284             self.x = state
285             self.p = covar
```

D.4.2 tracker.py

```
'''
    File name         : tracker.py
    File Description  : Tracker Using Kalman Filter &
↪   Hungarian Algorithm
    Author            : Srini Ananthakrishnan
    Date created      : 07/14/2017
    Date last modified: 07/16/2017
    Python Version    : 2.7
'''

# Import python libraries
import numpy as np
from kalman_filter_backup import KalmanFilter
from common import dprint
from scipy.optimize import linear_sum_assignment
from ukf import UKF
import time
import cv2
class Track(object):
    """Track class for every object to be tracked
    Attributes:
        None
    """

    def __init__(self, prediction, trackIdCount):
        """Initialize variables used by Track class
        Args:
            prediction: predicted centroids of object to
↪   be tracked
            trackIdCount: identification of each track
↪   object
        Return:
            None
        """
        self.track_id = trackIdCount  # identification of
        ↪   each track object
        self.KF = UKF(prediction)#,self.iterate_x)  # KF
        ↪   instance to track this object
```

```
34          self.prediction = np.asarray(prediction)  #
             ↪ predicted centroids (x,y)
35          self.skipped_frames = 0  # number of frames
             ↪ skipped undetected
36          self.trace = []  # trace path
37          #-----------ADDED BY VIRGINIA
             ↪ RAMÓN--------------------
38          self.bbox = []
39          self.obj_class = None
40
41  class Tracker(object):
42      """Tracker class that updates track vectors of object
         ↪ tracked
43      Attributes:
44          None
45      """
46
47      def __init__(self, dist_thresh, max_frames_to_skip,
         ↪ max_trace_length,
48                   trackIdCount):
49          """Initialize variable used by Tracker class
50          Args:
51              dist_thresh: distance threshold. When exceeds
    ↪ the threshold,
52                           track will be deleted and new
    ↪ track is created
53              max_frames_to_skip: maximum allowed frames to
    ↪ be skipped for
54                                  the track object undetected
55              max_trace_lenght: trace path history length
56              trackIdCount: identification of each track
    ↪ object
57          Return:
58              None
59          """
60          self.dist_thresh = dist_thresh
61          self.max_frames_to_skip = max_frames_to_skip
62          self.max_trace_length = max_trace_length
63          self.tracks = []
64          self.trackIdCount = trackIdCount
65          self.ls=[]
```

```
66          self.r = [2.5]
67      def Update(self, detections, bboxes, flag):
68          """Update tracks vector using following steps:
69              - Create tracks if no tracks vector found
70              - Calculate cost using sum of square distance
71                between predicted vs detected centroids
72              - Using Hungarian Algorithm assign the correct
73                detected measurements to predicted tracks
74
    ↪ https://en.wikipedia.org/wiki/Hungarian_algorithm
75              - Identify tracks with no assignment, if any
76              - If tracks are not detected for long time,
    ↪ remove them
77              - Now look for un_assigned detects
78              - Start new tracks
79              - Update KalmanFilter state, lastResults and
    ↪ tracks trace
80          Args:
81              detections: detected centroids of object to be
    ↪ tracked
82              bboxes: detected bounding boxes
83          Return:
84              None
85          """
86          assignment = []
87
88          if flag==1:
89              # Create tracks if no tracks vector found
90              if (len(self.tracks) == 0):
91                  for i in range(len(detections)):
92                      #print('detections : ',detections)
93                      track = Track(detections[i],
                         ↪ self.trackIdCount)
94                      # ------------------- ADDED BY
                         ↪ VIRGINIA RAMÓN
                         ↪ ------------------------
95                      track.bbox.append(bboxes[i])
96                      track.obj_class = bboxes[i][4]
97                      self.trackIdCount += 1
98                      self.tracks.append(track)
99
```

```
100             # Calculate cost using sum of square distance
                ↪  between
101             # predicted vs detected centroids
102             N = len(self.tracks)
103             M = len(detections)
104             cost = np.zeros(shape=(N, M))   # Cost matrix
105             #print ('N: ',N)
106             #print('M: ',M)
107
108             for i in range(len(self.tracks)):
109                 for j in range(len(detections)):
110                     try:
111                         # diff1 = np.subtract(self.tracks[
                            ↪  i].prediction[0] ,
                            ↪  detections[j][0])
112                         # diff2 = np.subtract(self.tracks[
                            ↪  i].prediction[1] ,
                            ↪  detections[j][1])
113                         # diff=np.array([[diff1],[diff2]])
114                         # distance =
                            ↪  np.sqrt(diff[0][0]*diff[0][0] +
115                                                 # diff[1][0
                                                    ↪  ]*diff[
                                                    ↪  1][0])
116                         # -------------------- ADDED BY
                            ↪  VIRGINIA RAMÓN
                            ↪  ------------------------
117                         # IoU: Get IoU of each BBox
                            ↪  (1-IoU) because we use
                            ↪  Hungarian Algorithm to assign
                            ↪  weights
118                         cost[i][j] =
                            ↪  1.0-bb_intersection_over_union
                            ↪  (self.tracks[i].bbox[-1],
                            ↪  bboxes[j],
                            ↪  self.tracks[i].obj_class)
119                         #print("differece:
                            ↪  ",diff,"prediction: ",self.tra
                            ↪  cks[i].prediction,"detections:
                            ↪  ",detections[j])
```

```
                    #print("cost: i: ",i," j: ",j,":
                     ↪  ",cost[i][j])
                except:
                    pass

        # Using Hungarian Algorithm assign the correct
         ↪  detected measurements
        # to predicted tracks
        #print(cost)
        for _ in range(N):
            assignment.append(-1)
        row_ind, col_ind = linear_sum_assignment(cost)
        #print(row_ind,col_ind)
        for i in range(len(row_ind)):
            assignment[row_ind[i]] = col_ind[i]
        # Identify tracks with no assignment, if any
        un_assigned_tracks = []
        for i in range(len(assignment)):
            if (assignment[i] != -1):
                # check for cost distance threshold.
                # If cost is very high then un_assign
                 ↪  (delete) the track
                if (cost[i][assignment[i]] >
                 ↪  self.dist_thresh):
                    assignment[i] = -1
                    un_assigned_tracks.append(i)
                pass
            else:
                self.tracks[i].skipped_frames += 1
        print ('Assignment after thresholding: ',
         ↪  assignment)
        # If tracks are not detected for long time,
         ↪  remove them
        del_tracks = []
        for i in range(len(self.tracks)):
            if (self.tracks[i].skipped_frames >
             ↪  self.max_frames_to_skip):
                del_tracks.append(i)
        if len(del_tracks) > 0:  # only when skipped
         ↪  frame exceeds max
            for id in del_tracks:
```

```
153                     if id < len(self.tracks):
154                         del self.tracks[id]
155                         del assignment[id]
156                     else:
157                         dprint("ERROR: id is greater than
                             ↪ length of tracks")
158 
159             # Now look for un_assigned detects
160             un_assigned_detects = []
161             for i in range(len(detections)):
162                     if i not in assignment:
163                         un_assigned_detects.append(i)
164             print('Unassigned detects:
                 ↪ ',un_assigned_detects)
165             # Start new tracks
166             if(len(un_assigned_detects) != 0):
167                 for i in range(len(un_assigned_detects)):
168                     track = Track(detections[un_assigned_d
                         ↪ etects[i]],
169                                   self.trackIdCount)
170                     self.trackIdCount += 1
171                     # -------------------- ADDED BY
                         ↪ VIRGINIA RAMÓN
                         ↪ ------------------------
172                     track.obj_class = bboxes[i][4]
173                     track.bbox.append(bboxes[un_assigned_d
                         ↪ etects[i]])
174                     self.tracks.append(track)
175 
176             # Update KalmanFilter state, lastResults and
                 ↪ tracks trace
177             for i in range(len(assignment)):
178                 #print('before only prediction: ',
                     ↪ self.tracks[i].KF.x)
179                 self.tracks[i].KF.predict(0.05)
180                 #print('after only prediction: ',
                     ↪ self.tracks[i].KF.x)
181                 if(assignment[i] != -1):
182                     self.tracks[i].skipped_frames = 0
```

```
183                     self.tracks[i].KF.update([0],np.array(
                         ↪  [detections[assignment[i]][0]]),1,
                         ↪  self.r)
184                     self.tracks[i].KF.update([1],np.array(
                         ↪  [detections[assignment[i]][1]]),1,
                         ↪  self.r)
185                     self.tracks[i].prediction=self.tracks[
                         ↪  i].KF.x
186                     #print("pre:
                         ↪  ",self.tracks[i].prediction)
187                     #self.tracks[i].prediction[1]=self.tra
                         ↪  cks[i].KF.x[1];
188                 else:
189                     #print(self.tracks[i].KF.x)
190                     self.tracks[i].KF.update([0],np.array(
                         ↪  np.array([0])),0,self.r)
191                     self.tracks[i].KF.update([1],np.array(
                         ↪  np.array([0])),0,self.r)
192                     self.tracks[i].prediction=self.tracks[
                         ↪  i].KF.x
193                     #self.tracks[i].prediction =
                         ↪  self.tracks[i].KF.correct(
                         #
194                          ↪  np.array([[0], [0]]), 0)
195                 #print('det
                     ↪  :',np.array([[detections[assignment[i]
                     ↪  ][0]],[detections[assignment[i]][1]]]))
196                 #print('i : ', assignment[i], 'updated
                     ↪  prediction :
                     ↪  ',self.tracks[i].prediction)
197                 if(len(self.tracks[i].trace) >
                     ↪  self.max_trace_length):
198                     for j in
                         ↪  range(len(self.tracks[i].trace) -
199                                  self.max_trace_length):
200                         del self.tracks[i].trace[j]
201
202
203                 if self.tracks[i].prediction.shape[0] == 2:
```

```
204                     self.tracks[i].prediction=np.insert(se
                         ↪  lf.tracks[i].prediction,self.track
                         ↪  s[i].prediction.shape[0],0)
205                     self.tracks[i].prediction=np.insert(se
                         ↪  lf.tracks[i].prediction,self.track
                         ↪  s[i].prediction.shape[0],0)
206                     self.tracks[i].prediction=np.insert(se
                         ↪  lf.tracks[i].prediction,self.track
                         ↪  s[i].prediction.shape[0],0)
207                     self.tracks[i].prediction=np.insert(se
                         ↪  lf.tracks[i].prediction,self.track
                         ↪  s[i].prediction.shape[0],0)
208
209                 self.tracks[i].trace.append(np.reshape(sel
                     ↪  f.tracks[i].prediction,(6,1)))
210                 #print('cc :
                     ↪  ',self.tracks[i].prediction.shape)
211                 self.tracks[i].KF.lastResult =
                     ↪  self.tracks[i].prediction
212                 self.ls = assignment
213                 # -------------------- ADDED BY VIRGINIA
                     ↪  RAMÓN ------------------------
214                 #print("Saving BBox...")
215                 #print(len(self.tracks[i].trace))
216                 if(assignment[i] != -1):
217                     #print("Not -1")
218                     self.tracks[i].bbox.append(bboxes[assi
                         ↪  gnment[i]])
219                 else:
220                     bbox = []
221                     #print(len(self.tracks[i].trace))
222                     if (len(self.tracks[i].trace) > 1):
223
224                         # Get last centroid
225                         x1 = self.tracks[i].trace[-2][0][0]
226                         y1 = self.tracks[i].trace[-2][1][0]
227
228                         x2 = self.tracks[i].trace[-1][0][0]
229                         y2 = self.tracks[i].trace[-1][1][0]
230
231                         xd = x2-x1
```

```
                        yd = y2-y1
                        # Center bbox previous to new
                         ↪  positiom
                        bbox = self.tracks[i].bbox[-1]

                        x1 = bbox[0]+xd if bbox[0]+xd > 0
                         ↪  else 0
                        y1 = bbox[1]+yd if bbox[1]+yd > 0
                         ↪  else 0
                        x2 = bbox[2]+xd if bbox[2]+xd > 0
                         ↪  else 0
                        y2 = bbox[3]+yd if bbox[3]+yd > 0
                         ↪  else 0

                        bbox = [x1,y1,x2,y2]
                    else:

                        bbox = self.tracks[i].bbox[-1]

                    self.tracks[i].bbox.append(bbox)


        else:
            assignment = self.ls
            for i in range(len(assignment)):
                #print('before only prediction: ',
                 ↪  self.tracks[i].KF.x)
                self.tracks[i].KF.predict(0.08)
                #print('after only prediction: ',
                 ↪  self.tracks[i].KF.x)

                self.tracks[i].KF.update([0],np.array(np.a⌋
                 ↪  rray([0])),0,self.r)
                self.tracks[i].KF.update([1],np.array(np.a⌋
                 ↪  rray([0])),0,self.r)
                self.tracks[i].prediction=self.tracks[i].K⌋
                 ↪  F.x
                    #self.tracks[i].prediction =
                     ↪  self.tracks[i].KF.correct(
                     #
                      ↪  np.array([[0], [0]]), 0)
```

```
261                 #print('det
                     ↪  :',np.array([[detections[assignment[i]
                     ↪  ][0]],[detections[assignment[i]][1]]]))
262                 #print('i : ', assignment[i], 'updated
                     ↪  prediction :
                     ↪  ',self.tracks[i].prediction)
263                 if(len(self.tracks[i].trace) >
                     ↪  self.max_trace_length):
264                     for j in
                         ↪  range(len(self.tracks[i].trace) -
265                                     self.max_trace_length):
266                         del self.tracks[i].trace[j]
267
268
269                 if self.tracks[i].prediction.shape[0] == 2:
270                     self.tracks[i].prediction=np.insert(se
                         ↪  lf.tracks[i].prediction,self.track
                         ↪  s[i].prediction.shape[0],0)
271                     self.tracks[i].prediction=np.insert(se
                         ↪  lf.tracks[i].prediction,self.track
                         ↪  s[i].prediction.shape[0],0)
272                     self.tracks[i].prediction=np.insert(se
                         ↪  lf.tracks[i].prediction,self.track
                         ↪  s[i].prediction.shape[0],0)
273                     self.tracks[i].prediction=np.insert(se
                         ↪  lf.tracks[i].prediction,self.track
                         ↪  s[i].prediction.shape[0],0)
274
275                 self.tracks[i].trace.append(np.reshape(sel
                     ↪  f.tracks[i].prediction,(6,1)))
276                 #print('cc :
                     ↪  ',self.tracks[i].prediction.shape)
277                 self.tracks[i].KF.lastResult =
                     ↪  self.tracks[i].prediction
278                 # -------------------- ADDED BY VIRGINIA
                     ↪  RAMÓN ------------------------
279                 #print("Saving BBox...")
280                 #print(len(self.tracks[i].trace))
281                 if(assignment[i] != -1):
282                     #print("Not -1")
```

```
283                     self.tracks[i].bbox.append(bboxes[assi
                         ↪ gnment[i]])
284                 else:
285                     bbox = []
286                     #print(len(self.tracks[i].trace))
287                     if (len(self.tracks[i].trace) > 1):
288 
289                         # Get last centroid
290                         x1 = self.tracks[i].trace[-2][0][0]
291                         y1 = self.tracks[i].trace[-2][1][0]
292 
293                         x2 = self.tracks[i].trace[-1][0][0]
294                         y2 = self.tracks[i].trace[-1][1][0]
295 
296                         xd = x2-x1
297                         yd = y2-y1
298                         # Center bbox previous to new
                             ↪ positiom
299                         bbox = self.tracks[i].bbox[-1]
300 
301                         x1 = bbox[0]+xd if bbox[0]+xd > 0
                             ↪ else 0
302                         y1 = bbox[1]+yd if bbox[1]+yd > 0
                             ↪ else 0
303                         x2 = bbox[2]+xd if bbox[2]+xd > 0
                             ↪ else 0
304                         y2 = bbox[3]+yd if bbox[3]+yd > 0
                             ↪ else 0
305 
306                         bbox = [x1,y1,x2,y2]
307                     else:
308 
309                         bbox = self.tracks[i].bbox[-1]
310 
311                     self.tracks[i].bbox.append(bbox)
312 
313 
314 # -------------------- ADDED BY VIRGINIA RAMÓN
    ↪ ------------------------
315 # Extracted from https://www.pyimagesearch.com/2016/11/07/
    ↪ intersection-over-union-iou-for-object-detection/
```

```
316 def bb_intersection_over_union(boxA, boxB, obj_class):
317     # determine the (x, y)-coordinates of the intersection
        ↪ rectangle
318     xA = max(boxA[0], boxB[0])
319     yA = max(boxA[1], boxB[1])
320     xB = min(boxA[2], boxB[2])
321     yB = min(boxA[3], boxB[3])
322     # compute the area of intersection rectangle
323     interArea = max(0, xB - xA + 1) * max(0, yB - yA + 1)
324     # compute the area of both the prediction and
        ↪ ground-truth
325     # rectangles
326     boxAArea = (boxA[2] - boxA[0] + 1) * (boxA[3] -
        ↪ boxA[1] + 1)
327     boxBArea = (boxB[2] - boxB[0] + 1) * (boxB[3] -
        ↪ boxB[1] + 1)
328     # compute the intersection over union by taking the
        ↪ intersection
329     # area and dividing it by the sum of prediction +
        ↪ ground-truth
330     # areas - the interesection area
331     iou = interArea / float(boxAArea + boxBArea -
        ↪ interArea)
332     # chwck if objects are the same class
333     if int(obj_class) != int(boxB[4]):
334         iou = -iou # to avoid being selected
335     # return the intersection over union value
336     return abs(iou)
```

D.4.3 object_tracking.py

```
'''
    File name         : object_tracking.py
    File Description  : Multi Object Tracker Using Kalman Filter
                        and Hungarian Algorithm
    Author            : Srini Ananthakrishnan
    Date created      : 07/14/2017
    Date last modified: 07/16/2017
    Python Version    : 2.7

    Modified by           : Virginia Ramón
    Data last modified: 19/12/2021
'''

# Import python libraries
import sys
import cv2
import copy
import numpy as np
import time
from tracker import Tracker

# --------------ADDED BY VIRGINIA RAMÓN----------------------
from frame_detection import frame_detection as vid
sys.path.append("/home/virginia/TFG/YOLOv4/TensorFlow-2.x-YOLOv3-master/")
from yolov3.utils import Load_Yolo_model
from yolov3.configs import *
# -----------------------------------------------------------

def main():
    """Main function for multi object tracking
    Usage:
        $ python2.7 objectTracking.py
    Pre-requisite:
        - Python2.7
        - Numpy
```

```
36          - SciPy
37          - Opencv 3.0 for Python
38      Args:
39          None
40      Return:
41          None
42      """
43
44
45      '''
46      options = {
47              'model': 'cfg/tiny-yolo-voc-1c.cfg',
48              'load': 4000,
49              'threshold': 0.15,
50              'gpu': 1.0
51      }
52
53      tfnet = TFNet(options)
54      '''
55      # --------------ADDED BY VIRGINIA
         ↪  RAMÓN----------------------
56      rg = 0
57      net = Load_Yolo_model()
58      # ------------------------------------------------------
         ↪  -------
59
60      # net = Detector(bytes("cfg/yolov3.cfg",
         ↪  encoding="utf-8"), bytes("weights/yolov3.weights",
         ↪  encoding="utf-8"), 0,
61                     # bytes("cfg/coco.data",
                        ↪  encoding="utf-8"))
62
63      # Create opencv video capture object
64      cap = cv2.VideoCapture("../IMAGES/GH010008_Val.mp4")
65      #Output path to save video result
66      output_path = "../IMAGES/GH010008_Val_UKF20000f.mp4"
67
68      cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1920)
69      cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 1080)
70
71      # by default VideoCapture returns float instead of int
```

```
    width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    fps = int(cap.get(cv2.CAP_PROP_FPS))
    codec = cv2.VideoWriter_fourcc(*'XVID')
    out = cv2.VideoWriter(output_path, codec, fps, (width, height)) # output_path must be .mp4

    # Create Object Tracker
    tracker = Tracker(0.75, 80, 10000, 40)

    # Variables initialization
    skip_frame_count = 0
    track_colors = [(255, 0, 0), (0, 255, 0), (0, 0, 255), (255, 255, 0),
                    (0, 255, 255), (255, 0, 255), (255, 127, 255),
                    (127, 0, 255), (127, 0, 127)]
    pause = False
    frame_array=[]
    first=0
    # Frames of track shown
    max_frames = 20000
    # Infinite loop to process video frames
    size=(1920,1080)
    c=0

    try:
        while True:
            # Capture frame-by-frame
            ret, frame = cap.read()

            # Make copy of original frame
            orig_frame = copy.copy(frame)

            # Skip initial frames that display logo
            '''
            if (skip_frame_count < 15):
                skip_frame_count += 1
                continue
            '''

```

```
110             # Detect and return centeroids of the objects
                ↪  in the frame
111             centers, bbxs = vid.detect(ret,frame,net)
112
113             # If centroids are detected then track them
114             if (len(centers) > 0):
115                 first=1
116                 # Track object using Kalman Filter
117                 tracker.Update(centers,bbxs,first)
118
119                 # For identified object tracks draw
                    ↪  tracking line
120                 # Use various colors to indicate different
                    ↪  track_id
121                 print('NUM OF OBJECTS :
                    ↪  ',len(tracker.tracks))
122                 if len(tracker.tracks) > 1:
123                     rg = len(tracker.tracks)
124                 else:
125                     rg = len(tracker.tracks)
126                 for i in range(rg):
127                     if (len(tracker.tracks[i].trace) > 1):
128                         # Check number of traces to show
129                         if (len(tracker.tracks[i].trace) >
                            ↪  max_frames):
130                             init_frames = max_frames
131                         else:
132                             init_frames = len(tracker.trac
                                ↪  ks[i].trace)
133
134                         for j in range(len(tracker.tracks[
                            ↪  i].trace)-init_frames,len(trac
                            ↪  ker.tracks[i].trace)-1):
135                             # Draw trace line
136                             x1 = tracker.tracks[i].trace[j
                                ↪  ][0][0]
137                             y1 = tracker.tracks[i].trace[j
                                ↪  ][1][0]
138                             x2 = tracker.tracks[i].trace[j
                                ↪  +1][0][0]
```

```
                        y2 = tracker.tracks[i].trace[j
                        ↪  +1][1][0]
                        clr =
                        ↪  tracker.tracks[i].track_id
                        ↪  % 9
                        cv2.line(frame, (int(x1),
                        ↪  int(y1)), (int(x2),
                        ↪  int(y2)),
                                track_colors[clr], 4)

                    xb1 = int(tracker.tracks[i].bbox[-
                    ↪  1][0])
                    yb1 = int(tracker.tracks[i].bbox[-
                    ↪  1][1])
                    xb2 = int(tracker.tracks[i].bbox[-
                    ↪  1][2])
                    yb2 = int(tracker.tracks[i].bbox[-
                    ↪  1][3])
                    cv2.rectangle(frame, (xb1,yb1),
                    ↪  (xb2,yb2), (0,0,255),1)
                    x = int(tracker.tracks[i].trace[-1
                    ↪  ][0][0])
                    y = int(tracker.tracks[i].trace[-1
                    ↪  ][1][0])
                    cv2.circle(frame, (x,y), 3,
                    ↪  (0,0,255),4)

            # Display the resulting tracking frame

        elif first==1:
            tracker.Update(centers,0)
            print('NUM OF OBJECTSno :
            ↪  ',len(tracker.tracks))
            if len(tracker.tracks) > 1:
                rg = len(tracker.tracks)
            else:
                rg = len(tracker.tracks)
            for i in range(rg):
                if (len(tracker.tracks[i].trace) > 1):
                    print('NUM OF OBJECTSnononono : ',l
                    ↪  en(tracker.tracks[i].trace),)
```

```
165                     print('trace :
                         ↪  ',tracker.tracks[i].trace[len(
                         ↪  tracker.tracks[i].trace)-1],)
166                     # Check number of traces to show
167                     if (len(tracker.tracks[i].trace) >
                         ↪  max_frames):
168                         init_frames = max_frames
169                     else:
170                         init_frames = len(tracker.trac
                             ↪  ks[i].trace)
171                     for j in range(len(tracker.tracks[
                         ↪  i].trace)-init_frames,len(trac
                         ↪  ker.tracks[i].trace)-1):
172                         # Draw trace line
173                         x1 = tracker.tracks[i].trace[j
                             ↪  ][0][0]
174                         y1 = tracker.tracks[i].trace[j
                             ↪  ][1][0]
175                         x2 = tracker.tracks[i].trace[j
                             ↪  +1][0][0]
176                         y2 = tracker.tracks[i].trace[j
                             ↪  +1][1][0]
177                         clr =
                             ↪  tracker.tracks[i].track_id
                             ↪  % 9
178
179                         cv2.line(frame, (int(x1),
                             ↪  int(y1)), (int(x2),
                             ↪  int(y2)),
180                                  track_colors[clr], 4)
181                     xb1 = int(tracker.tracks[i].bbox[-
                         ↪  1][0])
182                     yb1 = int(tracker.tracks[i].bbox[-
                         ↪  1][1])
183                     xb2 = int(tracker.tracks[i].bbox[-
                         ↪  1][2])
184                     yb2 = int(tracker.tracks[i].bbox[-
                         ↪  1][3])
185                     cv2.rectangle(frame, (xb1,yb1),
                         ↪  (xb2,yb2), (0,0,255),1)
```

```
                        x = int(tracker.tracks[i].trace[-1
                        ↪ ][0][0])
                        y = int(tracker.tracks[i].trace[-1
                        ↪ ][1][0])
                        cv2.circle(frame, (x,y), 3,
                        ↪ (0,0,255),4)


            height, width, layers = frame.shape
            size = (width,height)
            frame_array.append(frame)
            if output_path != '': out.write(frame)

            '''
            # Display the original frame
            #cv2.imshow('Original', orig_frame)
            if keyboard.is_pressed('q'):# 'q' key has been
    ↪ pressed, exit program.
                break
            # Slower the FPS
            cv2.waitKey(50)

            # Check for key strokes
            k = cv2.waitKey(50) & 0xff
            if k == 27:  # 'esc' key has been pressed, exit
    ↪ program.
                break
            if k == 112:  # 'p' has been pressed. this will
    ↪ pause/resume the code.
                pause = not pause
                if (pause is True):
                    print("Code is paused. Press 'p' to
    ↪ resume..")
                    while (pause is True):
                        # stay in this loop until
                        key = cv2.waitKey(30) & 0xff
                        if key == 112:
                            pause = False
                            print("Resume code..!!")
                            break
            '''
```

```
220             key = cv2.waitKey(1) & 0xFF
221
222             # Exit
223             if key == ord('q'):
224                 break
225
226             # Take screenshot
227             if key == ord('s'):
228                 cv2.imwrite('frame_{}.jpg'.format(time.time
                    ↪ ()),
                    ↪ frame)
229
230             c+=1
231     except:
232         print('Video Ended')
233     # When everything done, release the capture
234     finally:
235         #out = cv2.VideoWriter('result2.mp4',cv2.VideoWrite
            ↪ r_fourcc(*'MP4V'),
            ↪ int(cap.get(cv2.CAP_PROP_FPS)), size)
236         #print('11')
237         #for i in range(len(frame_array)):
238             # writing to a image array
239             #cv2.imshow('ff',frame_array[i])
240             # writing to a image array
241             #out.write(frame_array[i])
242         #out.release()
243
244         #out.release()
245         cap.release()
246         cv2.destroyAllWindows()
247
248
249 if __name__ == "__main__":
250     # execute main
251     main()
```

D.4.4 common.py

```
'''
    File name          : common.py
    File Description   : Common debug functions
    Author             : Srini Ananthakrishnan
    Date created       : 07/14/2017
    Date last modified: 07/14/2017
    Python Version     : 2.7
'''


def dprint(*args, **kwargs):
    """Debug print function using inbuilt print
    Args:
        args   : variable number of arguments
        kwargs : variable number of keyword argument
    Return:
        None.
    """
    # print(*args, **kwargs)
    pass
```

D.4.5 frame_detection.py

```
import time
import argparse
#from pydarknet import Detector, Image
import cv2
import sys
sys.path.append("/home/virginia/TFG/YOLOv4/TensorFlow-2.x-YOLOv3-master/")
from yolov3.utils import detect_frame
from yolov3.configs import *

class frame_detection:

    def detect(r,frame,net):

        average_time = 0
        centroids=[]
        bbx = []
        if r:
            start_time = time.time()

            # Predict Bboxes
            bboxes, classes = detect_frame(net, frame,
             ↪  input_size=YOLO_INPUT_SIZE, show=False,
             ↪  score_threshold=0.54, iou_threshold=0.45)

            end_time = time.time()
            average_time = average_time * 0.8 +
             ↪  (end_time-start_time) * 0.2

            print("FPS :", 1/(end_time-start_time),
             ↪  "Average_Time :", average_time, "BBXS
             ↪  :",len(bboxes))

            for b in bboxes:
                # draw bboxes in frame
                if b[5] < 1: # pigs bboxes different color
                    cv2.rectangle(frame,
                     ↪  (int(b[0]),int(b[1])),
                     ↪  (int(b[2]),int(b[3])), (255,0,0),2)
```

```
32                      cv2.putText(frame, 'Pig',
                         ↪  (int(b[0]),int(b[1])),
                         ↪  cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,
                         ↪  255, 0), 1, cv2.LINE_AA)
33                  else:
34                      cv2.rectangle(frame,
                         ↪  (int(b[0]),int(b[1])),
                         ↪  (int(b[2]),int(b[3])), (0,255,0),2)
35                      cv2.putText(frame, 'Person',
                         ↪  (int(b[0]),int(b[1])),
                         ↪  cv2.FONT_HERSHEY_SIMPLEX, 0.5,
                         ↪  (255,0,0), 1, cv2.LINE_AA)
36                  x = int(b[0]+((b[2]-b[0])/2))
37                  y = int(b[1]+((b[3]-b[1])/2))
38
39                  # save centtroid and bbox
40                  centroids.append((x,y))
41                  bbx.append((b[0],b[1],b[2],b[3],b[5]))
42
43          return centroids, bbx
```