UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Superior d'Enginyeries Industrial, Aeroespacial i Audiovisual de Terrassa

APLICACIÓ DE MACHINE LEARNING PER A LA DETECCIÓ I CLASSIFICACIÓ DE PRODUCTES AGRÍCOLES

Document:

Memòria

Autor:

Xavier Fresnedoso Moreno

Director /Co-director:

Jose Antonio Soria Pérez

Titulació:

Grau en Enginyeria en Tecnologies Industrials

Convocatòria:

Primavera

TREBALL FINAL D'ESTUDIS

Índex

Resum 4

Abstract 5

1 Introducció 6
1.1 Antecedents ... 6
1.2 Objecte ... 7
1.3 Abast ... 7
1.4 Requeriments ... 10
1.5 Motivació i justificació ... 11
1.6 Entregables i contingut de la memòria ... 11

2 Antecedents i/o revisió de l'estat de l'art 13
2.1 Perspectiva general ... 13
2.2 Que és el Machine Learning? ... 14
2.3 Aprenentatge supervisat ... 15
2.4 Que és el Deep Learning? ... 19
2.4.1 Backpropagation ... 20
2.5 Tipus de xarxes d'aprenentatge profund ... 22
2.5.1 Xarxes neuronals recurrents ... 24
2.5.2 Xarxes neuronals convolucionals ... 25
2.6 Eines per a la comunitat de Deep Learning ... 31

3 Implementació de l'algoritme 33
3.1 Visió general ... 33
3.2 Instal·lació del projecte i biblioteques ... 34
3.3 Realització i execució de l'algoritme de l'entrenament ... 35
3.4 Realització i execució de la inferència ... 43
3.5 Matriu de confusió ... 45

4 Resultats 47
4.1 Visió general ... 47
4.2 Índex d'error durant l'entrenament ... 47
4.3 Matriu de resultats ... 48
4.4 Pressupost ... 50
4.4.1 Cost d'enginyeria ... 51
4.4.2 Cost variable ... 51

4.4.3 Cost total 52

5 Conclusions i futures liníes de treball 53

5.1 Conclusions 53

5.2 Futures linies de treball 54

Agraïments 55

Bibliografia 56

Índex de figures

1.1 Exemple de màscara en una imatge realitzat amb l'eina LabelMe . . . 8
1.2 Exemple de segmentació semàntica en pomes i peres 10

2.1 Mencions historiques del Machine Learning publicades en llibres. Recurs: Google Ngram Viewer, 2021 14
2.2 Mètode d'aprenentatge supervisat per detectar fruites.....15
2.3 Nombre d'articles de Deep Learning en arXiv entre els anys 2015 i 2018. Recurs: arXiv, NESTA, 2019.....19
2.4 Esquema i equacions de la *retropropagació* 20
2.5 Perceptró.....22
2.6 Exemple d'un perceptró amb els corresponents valors de les entrades com dels pesos.....22
2.7 Funció d'activació on la sortida $y = 0$ quan $x < 0$, i $y = 1$ quan $x \geq 0$ 23
2.8 Xarxa *LSTM* on podem observar que conté neurones ocultes amb *loops* cap enrere en blau (aquest fer permet que puguin emmagatzemar informació en cel·les de memòria).....24
2.9 En una ConvNet, cada capa va aprenent diferents nivells d'abstracció. Amb multiples capes és pot conseguir identificar estructures més complexes en les dades d'entrada. Recurs: *DEEP LEARNING INTRODUCCIÓN PRÁCTICA CON KERAS*.....26
2.10 Fully Convolutional Network.....27
2.11 U-Net.....28
2.12 RCNN. Recurs: *https://arxiv.org/abs/1311.2524*.....28
2.13 Mask RCNN. Recurs: *Mask RCNN paper*.....30

3.1 Connectar Colab a Drive.....34
3.2 Instal·lació paquets Mask R-CNN 34
3.3 Instal·lació h5py 34
3.4 Instalació versió 1.x de TensorFlow i els *cocos*.....35
3.5 Importacions de les llibreries i fitxers.....36
3.6 Directori on es guarda la xarxa sense entrenar 36
3.7 Direccions de directoris, importacions de llibreries i altres funcions 37
3.8 Configuració per a l'entrenament del conjunt de dades Fruites 38
3.9 Funcions per afegir text en imatges.....38
3.10 Classe FruitesDataset.....39
3.11 Un altre metode de muntar el Colab al Drive 39
3.12 Preparació del dataset i acció de càrregar i mostrar imatges aleatories 40

3.13 Extensió de Tensorboard 40
3.14 Codi del model 41
3.15 Codi d'entrenament 42
3.16 Codi inferència 43
3.17 Obtenció de resultats 44
3.18 Codi Matriu de confusió 45
3.19 Codi Matriu de confusió 46

4.1 Error d'entrenament 47
4.2 Matriu de confusió 48
4.3 Exemples dels resultats obtinguts 49
4.4 Exemples dels resultats obtinguts 50
4.5 Exemples dels resultats obtinguts 50
4.6 Estimació del pressupost total 52

Índex de taules

2.1 Exemples de diferents models i les seves aplicacions 17

4.1 Desglossament dels costos d'enginyeria per a cada activitat 51
4.2 Cost variable 51

RESUM

En aquest treball es detalla el desenvolupament d'un algoritme de Machine Learning perquè mitjançant tècniques de processament d'imatges detecti i classifiqui productes agrícoles en fotografies. Per tant, es tracta d'un algoritme aplicat al sector agrícola que utilitza la tecnologia de la intel·ligència artificial.

Per a dur a terme aquest projecte, l'autor ha hagut d'extreure i preparar un conjunt de dades, imatges, que l'algoritme ha utilitzat per poder aprendre a detectar i classificar productes agrícoles. Aquestes imatges han estat extretes de bases de dades a través d'Internet i també realitzades per l'autor a través de càmera fotogràfica. El 70% del conjunt d'imatges s'ha utilitzat per a l'entrenament, el 20% per la prova i el 10% per la validació.

Aquest projecte inclou diferents etapes: obtenció i preparació de dades, selecció del model de predicció, entrenament, verificació i proves. L'algoritme ha estat creat a través d'un model anomenat *Mask R-CNN* que s'utilitza per a resoldre problemes de segmentació semàntica en aplicacions d'aprenentatge automàtic de visió per computadora i ha estat desenvolupat en codi Python. Per a la creació de l'algoritme s'ha utilitzat un servei de computació gratuït al núvol anomenat *Google Colab* que està associat a un compte de Gmail.

El model predictiu generat ha obtingut una exactitud del 60,25%. Concretament, una precisió en la detecció i classificació de pomes del 52,17% i en peres del 86,05%. Per a augmentar la precisió de l'algoritme de cara al futur és necessari entrenar-lo amb un nombre de dades més elevat.

ABSTRACT

In this essay is detailed the development of a Machine Learning algorithm to detect and classify agricultural commodities in photographs through image processing techniques. Therefore, this is about an algorithm applied to the agricultural sector, which use artificial intelligence technology.

To make this project, the author has had to extract and prepare a set of data, images, which are used by the algorithm to learn how to detect and classify agricultural products. These images have been extracted from databases through the Internet and taken by the author with a photo camera. Overall, the 70% of all pictures have been used to train the algorithm, 20% for testing and 10% for checking and validation.

This project includes different stages. Data collection and preparation, selection of prediction model, training, validation and testing. The algorithm has been created through a model called *Mask R-CNN,* which is used to solve semantic segmentation issues in automathic vision learning applications for computers and has been developed with Python code. For the algorithm creation has been used a free cloud computing service called *Google Colab,* which is associated with a Gmail account.

The generated predictive model has been obtained with an exactitude of 60,25%. Concretely, with a detection precision and classification of 52,17% for apples and 86,05% for pears. To increase the algorithm precision in the future, it is necessary to train it with a high data number.

1 Introducció

1.1 Antecedents

Al llarg de la història, s'han viscut tres Revolucions Industrials: l'inici de la mecanització, la línia de producció en massa i l'automatització de processos [1]. Actualment, estem vivint la quarta revolució anomenada indústria 4.0 de la qual se'n deriva el Smart Agriculture. La quarta revolució es caracteritza per l'enviament de dades a través d'Internet sense intervenció humana i, per aquest tipus d'aplicacions, la *Intel - ligència Artificial* (IA) , juga un paper clau en la implementació del Smart Agriculture.

Amb el pas dels anys, la IA, amb els algoritmes d'aprenentatge, s'ha convertit en un camp amb un extraordinari potencial per a desenvolupar, propiciant l'aparició d'infinites aplicacions que estan tenint un gran impacte en la societat fins al punt que estan canviant la manera de viure de les persones.

Els algoritmes de la IA poden agrupar-se en dues grans categories, una basada en regles o axiomes i una altra basada en l'aprenentatge semi-automàtic, també conegut com a *Machine Learning* (ML) i que permet que els algoritmes aprenguin a realitzar una tascar concreta sense haver sigut explícitament programats per realitzar-la.[2].

Als darrers anys hi ha hagut un gran avenç tecnològic pel que fa al camp de la IA, això fa que el moment actual sigui un dels de major desenvolupament tecnològic i és previsible que segueixi així en les pròximes dècades.[3]

La segmentació semàntica fa referència al procés d'enllaç de cada píxel en la imatge a una etiqueta o classe. Els sistemes de segmentació semàntica tenen com a objectiu delimitar els objectes de les diferents classes a escala de píxel a píxel, donant com a resultat qualsevol forma arbitraria.

El Smart Agriculture, persegueix millorar la qualitat dels cultius i dels productes conreats i alhora reduir l'impacte del medi ambient. Això s'aconsegueix mitjançant la recollida i anàlisis de dades dels cultius. Aquesta millora és possible gràcies a l'ús de noves tecnologies (drons, intel · ligència artificial, sensors, software, entre d'altres.) [4]. L'ús d'aquestes noves eines al sector agrícola facilita la tasca als agricultors i al mateix temps obtenir millor rendiment i qualitat.

L'ús del Deep Learning al sector agrícola mitjançant l'ús de prediccions, pot millorar la gestió de collites, reduir costos en tècniques d'R+D (recerca i desenvolupament), conèixer el clima que hi haurà en els pròxims anys, modificar decisions de producció

d'acord amb les tendències de consum i l'optimització de processos productius en temps real. Aquest, és un camp molt innovador dins de la indústria agrícola i en conseqüència molt poc desenvolupat; per tant, això és un motiu més per a desenvolupar aquest projecte [5].

1.2 Objecte

L'objectiu principal d'aquest projecte consisteix en aprendre a configurar i utilitzar un algoritme de *Machine Learning* per a què mitjançant tècniques de processament d'image detecti i classsifiqui productes agrícoles en fotografies.

1.3 Abast

Aquest projecte es focalitzarà en el sector primari, en l'àmbit de l'agricultura. Concretament, s'apliquen tècniques d'aprenentatge supervisat basades en *Deep Learning* (DL) per detectar i classificar productes agrícoles en fotografies, obtingudes tant de bases de dades disponibles a Internet com de càmera fotogràfica. En l'àmbit general, les tasques que cal realitzar per aplicar tècniques d'aprenentatge supervisat basades en *Deep Learning* són les següents:

- **1) Obtenció de les dades**

En aquesta s'obtenen les dades que els algoritmes de IA han de processar i que, a priori, serveixen per resoldre el problema a tractar. Aquestes dades, varien segons la tipologia de l'aplicació i poden ser desde vectors i matrius amb valors numèrics que representen xifres a caràcters alfanumèrics (codi ASCII) fins a imatges, com és l'objecte d'aquest treball.

- **2) Preparació**

Una vegada aconseguides les dades el següent pas consisteix a preparar-les perquè l'algoritme escollit al tercer pas les pugui processar. Operacions típiques en aquesta fase són: el *filtratge*, *etiquetatge* o *augmentació*.

El filtratge és una operació típica que consisteix a treure mostres i/o característiques del conjunt de dades que estadísticament es comporten com "outliers" i que poden condicionar el model i limitar la seva precisió, a posteriori. Aquest procés inclou des d'eliminar dades que s'hagin extret de sensors, analògics o digitals, o retocar paràmetres d'imatges - com ara contrast, brillantor, etc. - que ajudin a ressaltar característiques útils per ser detectades pel model.

A diferència de l'aprenentatge no supervisat on l'objectiu és trobar distribucions o classes d'entrada, l'aprenentatge supervisat requereix etiquetar cada mostra amb la classe corresponent perquè el model la pugui detectar. Aquí, l'estratègia va des d'utilitzar caràcters numèrics per representar cada classe (classe 1, classe 2, ..., classe

n), paraules representatives. En aplicacions d'imatge, l'etiquetatge sovint requereix l'ús de programari alternatiu per marcar el contorn d'objectes que apareixen en les instantànies mitjançant màscares (Fig. 1.1).

Figure 1.1: Exemple de màscara en una imatge realitzat amb l'eina LabelMe

Una altra operació que sovint es realitza en la preparació de dades quan no es disposen de moltes dades és l'augmentació (també conegut com a "*bootstrap*"). Aquesta operació consisteix a incrementar el nombre de mostres del conjunt inicial de manera artificial, però modificant algun paràmetre o característica d'entrada. En aplicacions de detecció d'objectes en imatges, per exemple, que generar imatges amb l'angle de referència modificat ajuda als models classificadors a generalitzar el problema i millorar la classificació.

Tant aquesta fase com l'anterior, sovint requereix un treball exhaustiu que, de vegades, suposa una despesa econòmica i temps considerables.

- **3) Conjunts d'entrenament i prova**

Amb les dades preparades, el següent pas és agrupar-les en dos grups: *Entrenament i Prova*. El primer conjunt serveix per configurar i adaptar el model perquè dugui a terme la tasca de classificació desitjada. D'aquí el nom "entrenament", ja que necessita tant les dades com les etiquetes amb les classes per tenir una referència de la desviació i actualitzar els seus paràmetres per corregir-los. En el conjunt de "test", les etiquetes s'utilitzen per determinar si la predicció del model és correcta. Un criteri habitual és el 80-20, que bàsicament consisteix a agafar el 80% de les dades inicials com a conjunt d'entrenament, i el 20% restant de test.

- **4) Creació del model**

Amb els conjunts d'entrenament, el següent pas és escollir el model (o algorisme de IA) que millor s'adapta per resoldre el problema. Existeixen moltes tècniques de classificació. Mètodes populars que funcionen amb dades numèriques i cadenes de paraules són: *k-Nearest Neighbors*, CART (*Classification and Regression Trees*), LVQ (*Learning Vector Quantization*), NN (*Neural Netowrks*) o SVM *(Support Vector Machines*) entre d'altres. Entre els algoritmes que s'utilitzen per imatge, darrerament s'han fet molt populars les xarxes CNN (*Covolutional Neural Networks*),

- **5) Entrenament**

En aquesta fase, el model utilitza les dades d'entrenament com a entrada per calcular la seva predicció (o valors de sortida) que identifiquen unívocament un determinat patró o classe d'entrada. Mitjançant les etiquetes, es determina l'error i, mitjançant algorismes d'optimització, es determina la correcció que requereixen els paràmetres del model.

- **6) Prediccions**

Amb els paràmetres del model configurats després de l'entrenament, el següent pas consisteix, a utilitzar el conjunt de test amb el model per realitzar prediccions. Aquí, l'objectiu de les etiquetes no és el de tornar a alterar el model com en el pas d'entrenament, sinó el d'obtenir noves prediccions per comparar-les amb l'etiquetatge realitzat al punt 2.

- **7) Avaluació i millora**

Si estadísticament el comportament del model no és el desitjat, el següent pas natural és revisar algun dels punts que intervenen en la configuració de les fases anteriors i tornar a repetir l'experiment.

Aquest projecte es planteja utilitzar tècniques de segmentació semàntica per la detecció i classificació de fruites en imatges fotogràfiques. Per l'exemple d'aquest projecte en qüestió s'utilitzen imatges de pomes i peres i s'exclouen d'altres productes agrícoles (Fig. 1.2). Aquest treball inclou el desenvolupament a nivell software dels scripts necessaris per a aquesta aplicació segons les fases en els punts anteriors. En aquest sentit, destacar que obtenir un determinat valor de precisió del model o la millora del mateix (punt 7) no formen part de l'objectiu d'aquest treball, si no que el treball es limita a mostrar el funcionament d'aquest.

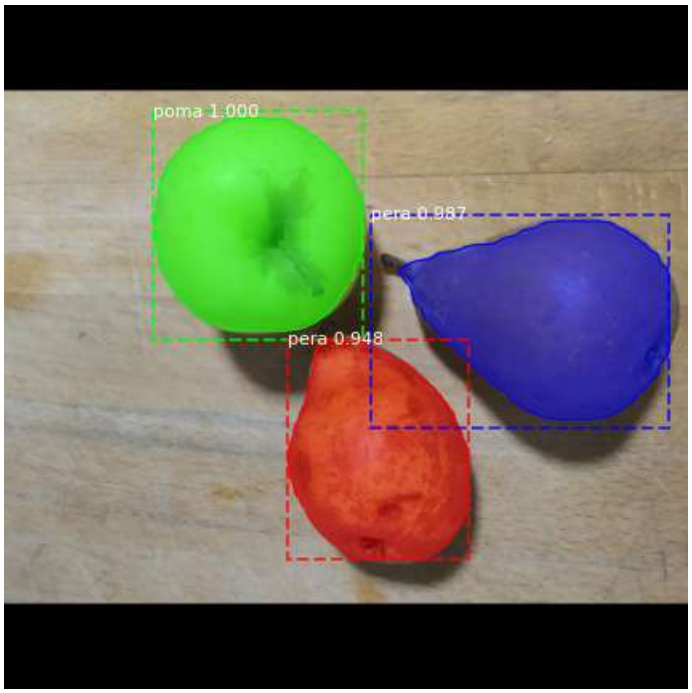

Figure 1.2: Exemple de segmentació semàntica en pomes i peres

1.4 Requeriments

Sota aquestes premisses, aquest projecte planteja l'ús d'eines amb les següents prestacions:

- **1) Bases de dades d'imatges**

S'ha agafat imatges de pomes i peres de bases de dades com ImageNet [6] , Kaggle [7] i COCO [8]. Les imatges són en color i amb una resolució variada en píxels (250 x 250, 450 x 480, 4900 x 4965...) en format JPEG (*Joint Photographic Expert Group*). Donat que el nombre d'imatges trobades no es considera suficient, també s'han realitzat instantànies amb càmera fotogràfica i aplicar tècniques de bootstrap. La càmera utilitzada és la càmera dual de 12+2 megapíxels amb IA del telèfon mòbil Xiaomi Redmi 7

S'han obtingut un total de 206 imatges que s'han separat en tres grups - entrenament, prova i validació - en una proporció de 70%, 20% i 10% respectivament.

- **2) Etiquetatge**

Per etiquetar les imatges s'ha utilitzat l'eina LabelMe [9]. Aquesta interfície està desenvolupada amb Python. A més d'obrir i mostrar les imatges, permet dibuixar els polígons amb els contorns, o màscares, dels objectes que es volen detectar i guardar-les dades en arxius de format JSON (*Java Script Object Notation*). Aquest format

permet etiquetar cada màscara dins de l'arxiu amb el nom de la classe corresponent. Per associar cada etiqueta amb la matge corresponent s'utilitza el mateix nom en tots dos arxius JPEG i JSON.

- **3) Model de predicció**

Com a model per a la classificació i detecció de pomes i peres s'ha utilitzat Mask-RCNN (*Mask Recursive Convolutional Neural Network*). Mask-RCNN és una xarxa neuronal profunda destinada a resoldre problemes de segmentació semàntica en aplicacions d'aprenentatge automàtic de visió per computador [10]. Les llibreries per treballar amb aquest model s'han desenvolupat en llenguatge Python i estan disponibles a Matterport [11]]. Per la qual cosa es poden utilitzar en diferents plataformes Linux, Windows i MAC.

- **4) Entrenament i Inferència**

Tots dos scripts per l'entrenament i la inferència s'han executat sobre la plataforma *GoogleColaboratory* [12] . Donat que l'entrenament és el script que més temps de processament i recursos computacionals requereix, s'ha decidit aquesta opció per córrer els algorismes en els servidors de Google, que contenen GPUs (Graphic Processor Units) per accelerar l'execució. Això requereix pujar totes les imatges preparades al punt 1 a la unitat de *Google Drive* associada al mateix compte de Gmail.

- **5) Precissió del model**

Per mostrar els resultats s'utilitza la matriu de confusió *(Confussion Matrix*) per indicar tant els encerts del model en predir la classe de les etiquetes com les errades.

La precisió obtinguda amb l'algoritme de Mask_RCNN ha estat d'un 60,25% d'encert.

1.5 Motivació i justificació

Aquest projecte es desenvolupa amb la finalitat de conèixer les tècniques i l'ús del Deep Learning i així poder crear un algoritme amb el qual es pugui contribuir en la millora del sector agrícola i en conseqüència aportar un granet de sorra per reduir l'impacte ambiental que té aquest sector.

1.6 Entregables i contingut de la memòria

En aquest projecte, es duran a terme els lliuraments següents per poder assolir els objectius proposats:

- Data:implica el conjunt de les imatges i de les etiquetes necessàries per a poder entrenar i executar l'algoritme.

- Sofware: implica els scripts desenvolupats de l'algoritme així com la xarxa entrenada.

El contingut de la memòria està estructurat en cinc capítols. En aquest primer, s'han preorientat les directrius generals i els objectius d'aquest treball. El segon, presenta els conceptes principals del Machine Learning i l'estat de l'art. També es parla dels dos tipus de xarxes més importants d'aprenentatge profund que hi ha, així com les eines que més s'utilitzen dins la comunitat de DL. El tercer, explica detalladament el desenvolupament dels scripts de l'entrenament i de la inferència. El quart, mostra i explica els resultats obtinguts i comportament de l'algoritme i també es detalla el pressupost del projecte.Finalment, el cinqué analitza quins haurien de ser els propers passos per millorar l'algoritme actual. Això considera l'obtenció de més imatges de les classes desenvolupades, així com l'obtenció -d'imatges de productes agrícoles nous i en conseqüència la modificació de l'algoritme.

2 Antecedents i/o revisió de l'estat de l'art

2.1 Perspectiva general

Des de la Revolució Industrial, fins als dies d'avui les màquines han recorregut un llarg camí. Aquestes, continuen omplint les fàbriques industrials, negocis i llars de tot el món, però ara les funcions que desenvolupen s'estenen molt més enllà d'activitats manuals o tasques cognitives que fins fa poc temps, només els humans eren capaços de realitzar. Conduir automòbils, guanyar a escacs als humans o fer feines de la llar, són alguns dels exemples actuals de tasques que les màquines saben dur a terme.

Aquests avenços, però, també causen alguns temors, com per exemple l'amenaça de la seguretat laboral. Segons un estudi de la British Broadcasting Company (BBC), professionals com cambrers, treballadors de bars, recepcionista o conductor de taxi tenen una alta probabilitat que el seu lloc de treball s'automatitzi cap a l'any 2035[13].

Els científics de dades i els enginyers de *Machine L*earning (ML), són els encarregats d'administrar i supervisar els algoritmes estadístics, les màquines es basen en aquests algoritmes per a poder funcionar. Per tant, l'estadística clàssica és un dels dos pilars fonamentals per a poder arribar a construir i programar aquestes màquines. Els algoritmes derivats d'aquesta branca de les matemàtiques són metafòricament parlant les cèl · lules sanguínies i l'oxigen que impulsen el ML. L'altre pilar fonamental és la programació informàtica[14].

2.2 Que és el Machine Learning?

El *Machine Learning* o *Aprenentatge Automàtic*, és una branca de la intel·ligència artificial, on el seu propòsit general és dotar a les màquines de l'habilitat perquè puguin aprendre per si mateixes, sense la necessitat que ningú perfeccioni els seus algoritmes [15].Una habilitat essencial per fer sistemes que no només siguin intel·ligents, sinó autònoms i capaços d'identificar patrons a les dades per convertir-los en prediccions [14]. D'aquesta manera les màquines passen d'executar tasques senzilles a unes altres molt més complexes.

L'any 1950, Alan Turing es va plantejar la pregunta de si és possible que les màquines pensin. Una mica més tard, al llarg de la dècada dels cinquanta, diversos científics van començar a investigar com aplicar la biologia de la xarxa neuronal del cervell humà per a crear màquines intel·ligents. [16]. Aquest va ser l'inici, on el *Machine Learning* va començar a evolucionar.

En el següent gràfic, es pot observar com no va ser fins a la dècada dels anys cinquanta quan es van començar a fer mencions sobre el *Machine Learning* a llibres. El 1959 hi havia un 0,0000016% de mencions del *Aprenentatge Automàtic* en llibres, en canvi, el 2019 aquest percentatge ja havia augmentat fins a un 0,0002781%. Aquest gràfic afirma la tendència de l'interès pel ML dels darrers anys. Actualment, aquesta és una de les àrees d'investigació més actives de la intel·ligència artificial.

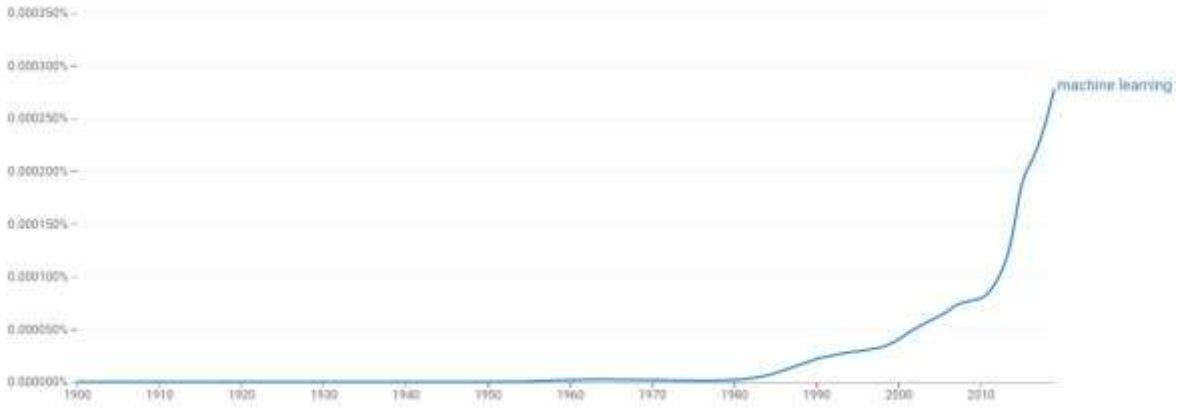

Figura 2.1: Mencions historiques del Machine Learning publicades en llibres. Recurs: Google Ngram Viewer, 2021

2.3 Aprenentatge supervisat

L'aprenentatge supervisat és la forma més comú de *Machine Learning*, on els algoritmes utilitzen dades d'imatges que han estat prèviament etiquetades, amb l'objectiu de trobar una funció que (depenent de les dades d'entrada) els hi assigni l'etiqueta de sortida que sigui adequada [17].

Per exemple, volem desenvolupar un algoritme que sigui capaç de classificar imatges que contenen diverses fruites, pomes, peres, llimones, síndria, taronges, plàtans, etc. El primer que hem de fer és recopilar un gran conjunt d'imatges de les fruites que volem detectar, cadascuna etiquetada en la categoria corresponent. Durant l'entrenament, a la màquina se li mostra una imatge, aquesta produeix una sortida en forma de vector de puntuacions (un vector per a cada categoria). L'objectiu és que la categoria desitjada obtingui la puntuació més elevada de totes les categories. Seguidament es calcula una funció objectiva que mesura l'error entre les puntuacions de sortida i el patró desitjat de puntuació. Un cop calculada la funció de sortida, la màquina modifica els paràmetres interns ajustables (pesos) per reduir l'error.

Per ajustar correctament el vector del pes, l'algoritme d'aprenentatge calcula un vector de gradient que, per a cada pes, indica en quina quantitat augmentaria o disminuiria l'error si s'incrementés el pes lleugerament.

Un cop acabat l'entrenament, s'analitza el rendiment del sistema en un conjunt de prova. Aquest serveix per a analitzar la capacitat de generalització de la màquina, és a dir, examinar la seva capacitat de produir respostes sensates quan analitza imatges que mai ha vist [18].

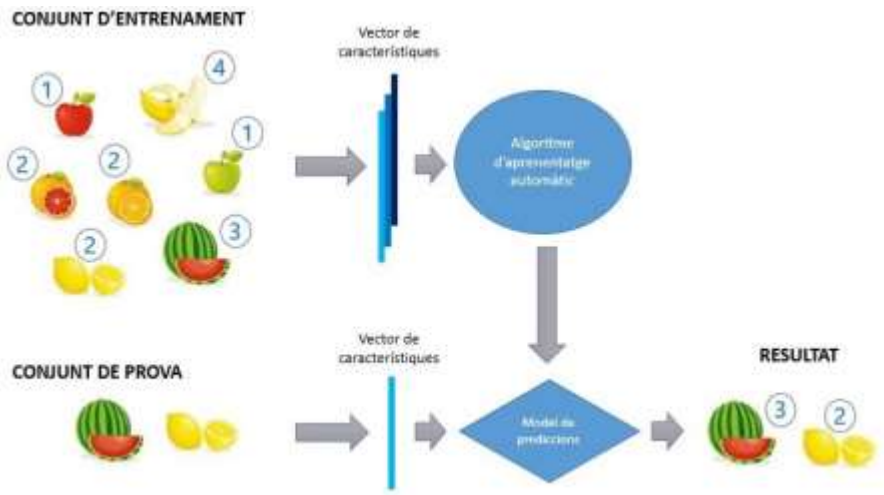

Figure 2.2: Mètode d'aprenentatge supervisat per detectar fruites

En resum, podem dir que l'objectiu del *Machine Learning* és derivar el significat de les dades, per tant, aquestes són la clau per a desbloquejar el ML. L'*Aprenentatge*

Automàtic està compost per set passos i cada un d'ells gira entorn de les dades. Seguidament es descriurà cadascun dels passos a seguir per a poder arribar a construir l'algoritme; en aquest cas un que sigui capaç de detectar productes agrícoles.

1. **Recollida de dades**: El ML requereix analitzar moltes dades d'informació a partir d'imatges sobre aquella temàtica la qual vulguem fer l'algoritme. En el meu cas la temàtica són els productes agrícoles, concretament es vol crear un algoritme que detecti pomes i peres. Per tant, aquest pas consisteix a recopilar un seguit d'imatges on apareguin aquests dos productes, ja sigui imatges on només aparegui un dels dos productes o bé els dos. La recollida de les imatges s'agruparà en tres carpetes (entrenament, prova i validació).

   La qualitat i la quantitat d'informació recopilada té un pes molt important en l'algoritme, ja que aquesta tindrà un impacte favorable o perjudicial en el funcionament del model.

   Les imatges de validació, representen entre el 70 % i el 80 % de les dades totals. Aquestes han estat extretes de bases de dades de productes agrícoles com *Kaggle* i com el seu nom diu serveixen per entrenar a l'algoritme. L'altre 20 % - 30 %, són imatges de prova extretes per mi mateix a través d'una càmera fotogràfica, són imatges que mai ha vist la màquina i s'utilitzen per provar l'algoritme. Tant les imatges d'entrenament com les de prova, han de tenir totes les fruites que volem que detecti l'algoritme etiquetades. Aquest etiquetatge es farà amb el programa *labelme*.

   Finalment tenim les imatges de validació, aquestes serveixen per comprovar com actua la xarxa neuronal (aquestes imatges no han de tenir cap etiqueta i no hem de tenir un nombre concret d'aquestes dades).

2. **Netejar les dades:** Les dades brutes no són molt útils, per aquest motiu cal preparar les imatges, normalitzar-les, eliminar aquelles que estan duplicades, corregir errors, gestionar valors que falten, etc. Per tant, aquest punt consisteix a supervisar que no hi hagi cap dada duplicada i que totes les imatges estiguin amb la seva corresponent etiqueta.

3. **Dividir les dades en imatges d'entrenament, prova i validació:** Aquest punt si bé no ho hem fet anteriorment, consisteix a separar les imatges en dues carpetes diferents. A la carpeta d'entrenament es trobaran totes aquelles imatges extretes de les bases de dades amb les seves etiquetes; per una altra banda, a la carpeta de prova hi haurà totes les imatges realitzades amb càmera fotogràfica i amb les seves etiquetes corresponent, i finalment a la carpeta de validació s'hi trobaran la resta de dades que no estan marcades.

4. **Crear un model:** Arribats a aquest punt, és l'hora de triar el model que més s'ajusti al nostre objectiu. Existeixen un seguit de models, cadascun d'ells han estat creats per una aplicació concreta. Hi ha algoritmes de classificació, predicció, regressió lineal, bayesià, etc. i hi ha variants d'aquests en funció del tipus de dades a processar (imatges, so, text, valors numèrics, etc.). Seguidament es mostren alguns d'aquests models i les seves aplicacions:

Model	Aplicacions
Logistic Regression	Predicció de preus d'inmobles
FCN	Classificació
CNN	Procesament d'imatges per trobar objectes en fotos
RNN	Reconeixement de veu
Mask R-CNN	Genera caixes delimitadores i màscares de segmentació per a cada instància d'un objecte de la imatge
Random Forest	Detecció de frau
Reinforcement Learning	Aplicació per ensenyar-li a la màquina a jugar a videojocs i guanyar
Generative Models	Creació d'imatges
K-means	Segmentar audiencies o inventaris
Classificadors Bayesians	Classificació d'emails: si aquests són spam o no

Table 2.1: Exemples de diferents models i les seves aplicacions

Per a la detecció i classificació de productes agrícoles utilitzaré la Mask R-CNN, aquesta està basada en la FPN i en una columna vertebral anomenada ResNet101.

5. **Entrenar el model:** Aquest pas consisteix a entrenar el model escollit. En el meu cas, s'utilitzaran les dades d'entrenament extretes de bases de dades d'internet per a executar la nostra màquina; en aquesta etapa d'entrenament s'ha de poder veure una millora incremental en la predicció del model. Inicialment els pesos del model tindran el valor "coco". Els "cocos" són uns pesos ja definits per investigadors per a poder realitzar models de reconeixement d'objectes. Aquests pesos han estat definits després d'haver processat milers d'imatges amb milions d'objectes etiquetats. Per tant, aquests pesos ens serveixen de punt de partida perquè l'algoritme trobi els nostres pesos determinats (si no utilitzéssim els cocos, hauríem de processar milers d'imatges per a poder arribar a obtenir valors semblants en els pesos) a mesura que l'algoritme es va entrenant.

6. **Fer les prediccions:** El següent pas que toca fer és realitzar les prediccions o inferència. És a dir, provar el model utilitzant unes dades completament noves

(imatges que mai ha vist l'algoritme). L'algoritme ens donarà per sortida les imatges amb les etiquetes i les màscares corresponents.

7. **Evaluar i millorar les prediccions:** Arribats a aquest punt el que cal fer és avaluar els resultats obtinguts. És a dir, comprovar si l'algoritme ha fet les prediccions correctament i si no ha estat així, s'ha de reajustar els paràmetres o utilitzar més dades per a poder solucionar aquest problema. Un cop s'han redefinit tots els paràmetres, s'haurà de tornar a entrenar el model.

2.4 Que és el Deep Learning?

El *Deep Learning* (acronim en anglès DL) o *Aprenentatge Profund*, és una branca del *Machine Learning* que està gairebé totalment inspirat en el aprenentatge supervisat. El DL permet que models computacionals formats per múltiples capes de processament siguin capaços d'aprendre representacions de dades amb diversos nivells d'abstracció. Amb el pas del temps aquests mètodes han millorat dràsticament l'estat de la tècnica en el reconeixement de veus, reconeixement visual i detecció d'objectes, entre d'altres. El *Deep Learning* descobreix una estructura intrincada en grans conjunts de dades mitjançant l'ús de l'algoritme de retropropagació en la capa anterior. Les xarxes convolucionals profundes han estat les culpables de grans avenços en el processament d'imatges, vídeo, veu i àudio [18].

L'*Aprenentatge Profund* és una branca que està en creixement i que s'ha possat molt de moda en els darrers anys, això ho podem verificar analitzant el nombre d'articles que s'han publicat respecte a aquesta temàtica en els darrers anys. En el següents gràfic, podem observar com l'interès per aquest mètode d'aprenentatge automàtic està creixent en tots els continents; observem com la regió amb major nombre d'articles publicats entorn de l'*Aprenentatge Profund* va ser Amèrica del Nord, seguit d'Europa [19] .

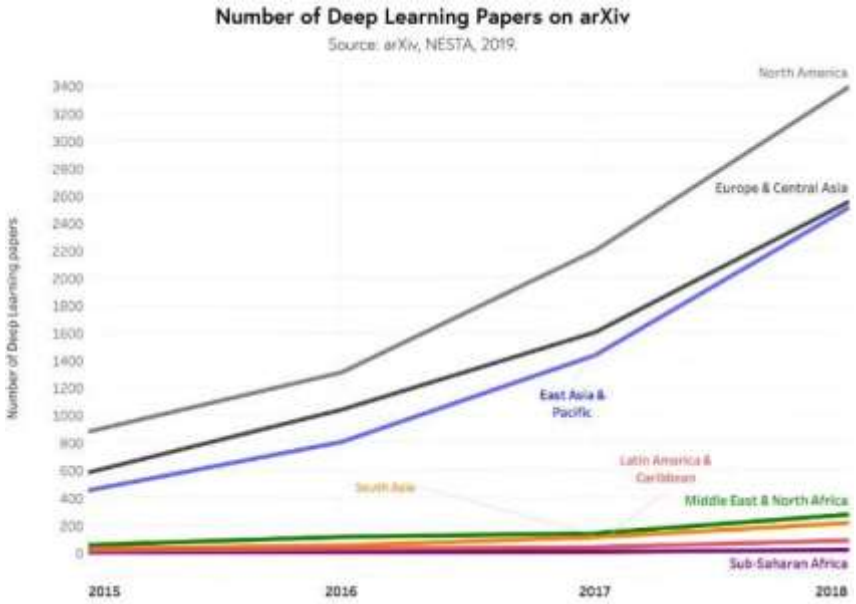

Figura 2.3: Nombre d'articles de Deep Learning en arXiv entre els anys 2015 i 2018. Recurs: arXiv, NESTA, 2019

L'aprenentatge de representacions és un conjunt de mètodes que permet a un sistema descobrir automàticament les representacions necessàries per a la detecció o classificació de característiques a partir de dades sense processar [20]. El *Deep Learning* són mètodes d'aprenentatge de representacions amb diversos nivells de representació,

obtinguts en formar mòduls simples, però no lineals que transformen la representació en un nivell (començant per l'entrada sense processar) en una representació de nivell superior, lleugerament més abstracta. Per a tasques de classificació, les capes superiors de representació amplifiquen aspectes d'entrada que són importants per a la discriminació i eliminen aquelles variacions considerades irrellevants. L'aspecte clau de l'*Aprenentatge Profund* és que aquestes capes de característiques no han estat creades per humans, sinó que aprenen de les dades per si mateixes[18].

L'arquitectura de l'*Aprenentatge Profund* és una pila de múltiples capes de mòduls simples, dels quals tots o la majoria depenen de l'aprenentatge i molts dels quals calculen assignacions d'entrada-sortida no lineals.

2.4.1 Backpropagation

La *backpropagation* o en català *retropropagació* és el mètode habitual que s'utilitza per entrenar arquitectures multicapa.

Des dels primers dies del reconeixement de patrons (1955) l'objectiu de la investigació era substituir les funcions realitzades a mà per xarxes multicapa entrenables, però malgrat la seva senzillesa, la solució no es va entendre fins a mitjans de la dècada de 1980. Les arquitectures multicapa es poden entrenar mitjançant un simple descens del gradient estocàstic. Sempre que els mòduls siguin funcions relativament suaus de les seves entrades i dels pesos interns, es poden calcular gradients usant el procediment de *retropropagació*. El procediment per calcular el gradient d'una funció objectiu respecte als pesos d'una pila de mòduls multicapa no és res més que una aplicació del concepte matemàtic de la regla de la cadena per a derivades. La idea clau és que el gradient de l'objectiu respecte a l'entrada d'un mòdul es pot calcular treballant cap enrere des del gradient respecte a la sortida d'aquest mòdul.

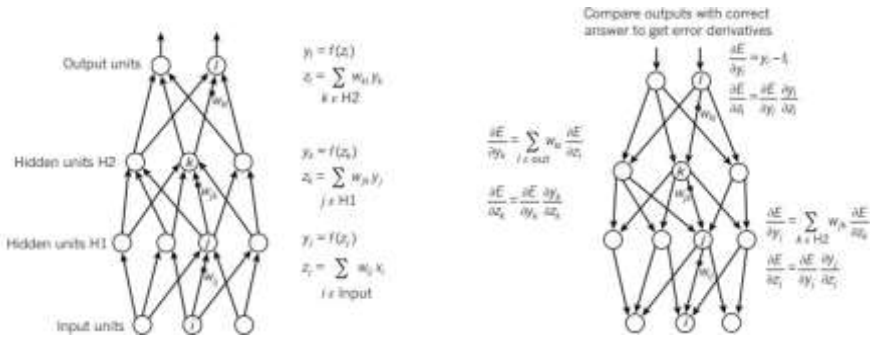

Figure 2.4: Esquema i equacions de la *retropropagació*

Aquesta equació es pot repetir per a propagar els gradients a través de tots els mòduls, començant des de la sortida (on la xarxa fa la seva predicció) fins a la

part inferior (on s'alimenta l'entrada externa). Per passar d'una capa a la següent, un conjunt d'unitats calcula una suma ponderada de les seves entrades de la capa anterior i passa el resultat a través d'una funció no lineal. Actualment, la funció no lineal més popular és la unitat lineal rectificada (ReLU), que és simplement el rectificador de mitja ona $f(z) = max(z, 0)$.

A les darreres dècades, les xarxes neuronals van fer servir no linealitats més suaus, com $tanh(z)$ o $\frac{1}{1+e^{-z}}$. Normalment, ReLU aprèn molt més ràpid en xarxes multicapa, això permet l'entrenament d'una xarxa supervisada profunda sense entrenament previ no supervisat.

A finals de la dècada del 1990, les xarxes neuronals i la *retropropagació* van ser abandonades i ignorades per la comunitat de DL; es creia que el descens del gradient simple quedaria atrapat en mínims locals, configuracions de pesos pels quals cap petit canvi reduiria l'error mitjà.

No va ser fins a l'any 2006 quan l'interès per aquestes xarxes de retroalimentació va ressorgir gràcies a un grup d'investigadors (CIFAR). Aquests, van introduir procediments d'aprenentatge no supervisat que podien crear capes de detectors de característiques sense haver de tenir dades etiquetades. L'objectiu d'aprendre cada capa de detectors de característiques era poder reconstruir o modelar les activitats de les entrades sense processar a la capa següent. Mitjançant l'entrenament previ de diverses capes de detectors de característiques progressivament més complexes, els pesos d'una xarxa profunda podrien inicialitzar-se a valors sensibles. Després es podria afegir una capa final d'unitats de sortida a la part superior de la xarxa i tot el sistema profund podria ajustar-se utilitzant la *backpropagation*.

La primera aplicació important d'aquest enfocament de preentrenament va ser el reconeixement de veu i va ser gràcies a l'aparició d'unitats de processament gràfic ràpid, les GPUs (xarxes molt còmodes de programar i que permetien crear xarxes 10 o 20 cops més ràpidament). El 2009, aquest descobriment es va utilitzar per a enllaçar curtes finestres temporals de coeficients extrets d'una ona sonora a un conjunt de probabilitats pels diversos fragments de parla que podrien representar el marc al centre de la finestra. L'any 2012, molts grups de parla van desenvolupar versions del descobriment del 2009 i les van aplicar als telèfons Android. [18].

2.5 Tipus de xarxes d'aprenentatge profund

L'any 1958, el psicòleg Frank Rosenblatt inspirat en el treball de Warren McCulloch i Walter Pitts va crear el Perceptró, la unitat des d'on naixerien i es potenciarien les xarxes neuronals artificials.

El perceptró de Rosenblatt consisteix en una o més entrades binàries (*x1, x2, x3…)*, un processador i una única sortida (*y*). Per a calcular la sortida aquest psicòleg va introduir el concepte de *"pesos" (w1, w2, w3…),* un nombre real que expressa la importància de la respectiva entrada amb la sortida. La sortida de la neurona serà 1 o 0 si la suma de la multiplicació de pesos per entrades és major o menor a un determinat valor llindar. En aquella època tant el valor dels *pesos* com el *valor llindar,* estaven assignats manualment pel científic.

Anys més tard de crear-se el perceptró, es va crear el perceptró multicapa; aquest només va ser una simple ampliació del perceptró passant d'una única neurona a més d'una. A mesura que hi havia més perceptrons a les capes, era més difícil de trobar els pesos per obtenir les sortides desitjades [21].

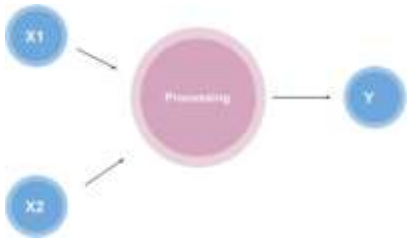

Figure 2.5: Perceptró

Seguidament s'explicarà el funcionament del perceptró a partir d'un exemple. Tenim un perceptró format per dues entrades, entrada 1: 3x = 24 amb un pes de 0,5 i entrada 2: 2x16 amb un pes de -1,0. El següent pas consisteix a multiplicar cada entrada pel seu pes corresponent (entrada 1: $24 \times 0,5 = 12$ i entrada 2: $16 \times (-1,0) = -16$).

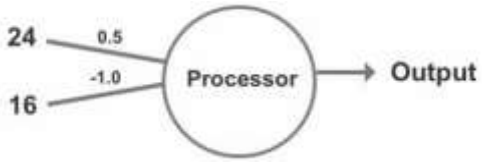

Figure 2.6: Exemple d'un perceptró amb els corresponents valors de les entrades com dels pesos.

Passant la suma dels pesos per la funció d'activació es genera la sortida del perceptró. Una característica clau del perceptró és que només és capaç de registrar dos possibles resultats, "1" o "0". El valor "1" activa la funció d'activació, en canvi, el "0" no. Tot i que el perceptró té naturalesa binària, hi ha diverses maneres per configurar la funció d'activació. En aquest exemple, hem fet que la funció d'activació ha de ser ≥ 0.Per tant, si la suma és major o igual a zero, la sortida és 1. En canvi, si la suma té un valor negatiu, la sortida serà 0.

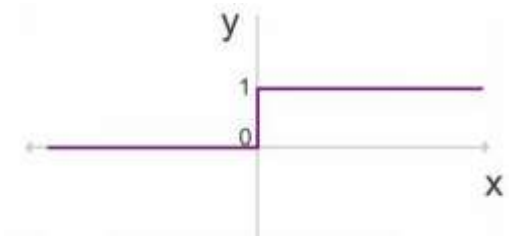

Figure 2.7: Funció d'activació on la sortida $y = 0$ quan $x < 0$, i $y = 1$ quan $x \geq 0$

Seguint amb el nostre exemple si sumem els productes de les entrades pels pesos obtenim el següent $12 + (-16) = -4$, per tant, amb aquest valor no es podrà activar la funció d'activació del perceptró.

Quan es treballa amb un model més gran de capes de xarxa neuronal, es configurarà el valor de "1" per poder passar la sortida a la següent capa. Per contra, el valor "0" està configurat per a ignorar-se i així impedir passar a la capa següent.

En l'*Aprenentatge Supervisat* els perceptrons es poden usar per formar dades i desenvolupar models de prediccions. Els passos per a la formació de dades són els següents:

1. S'introdueixen les entrades als processadors (neurones i nodes).
2. El perceptró calcula el valor de les entrades.
3. El perceptró calcula l'error entre l'estimació i el valor real.
4. El perceptró ajusta els seus pesos segons l'error.
5. Es repeteixen els passos anteriors fins que la persona que està duent a terme l'algoritme està satisfet amb la precisió del model.

La debilitat del perceptró és que en tenir una sortida binària, petits canvis en els pesos o biaixos en qualsevol perceptró dins d'una xarxa neuronal més gran pot induir a la polarització de resultats. Això pot conduir a canvis inesperats a la xarxa i a un gir total respecte a la sortida final. Per tant, això fa que sigui molt difícil formar un model precís que es pugui aplicar amb èxit a les dades de prova i a les entrades de dades futures. Per aquest motiu es van crear alternatives com per exemple la neurona sigmoide, capaç d'acceptar qualsevol valor entre 0 i 1.[14]

A continuació, s'explicaran quines són les dues principals xarxes neuronals que s'utilitzen al Deep Learning.

2.5.1 Xarxes neuronals recurrents

Les *Xarxes Neuronals Recurrents* o en anglès *Recurrent Neural Networks* (conegudes amb l'acrònim *RNN*), són un tipus de xarxa que s'utilitzen per analitzar dades de sèries temporals permetent tractar la dimensió del temps. Quan es va introduir la backpropagation, l'ús més interessant va ser entrenar aquest tipus de xarxes neuronals. Les *RNN* són molt potents per tot allò que té a veure amb l'anàlisi de seqüències, és a dir, anàlisi de text, so o vídeo. Aquestes, van ser creades a la dècada de 1980. El hàndicap que tenen és que aquestes xarxes han estat molt difícils d'entrenar, ja que els gradients de *retropropagació* o bé creixen o bé es redueixen amb el pas del temps, de manera que a mesura que passa el temps normalment aquests gradients desapareixen. Seguidament s'explicaran les *RNN* més conegudes:

- LSTM: són un tipus de xarxa que resolen la problemàtica a l'entrenament que tenien les xarxes neuronals recurrents convencionals (recordem que aquesta problemàtica era deguda al fet que els gradients tendien a créixer o a decréixer amb el pas del temps). Les *LSTM* resolen aquest problema gràcies al fet que incorporen un seguit de passos per decidir quina informació serà emmagatzemada i quina suprimida. La unitat de memòria *LSTM* conté tres portes que controlen la manera en què la informació flueix per la unitat. Aquest tipus de xarxes s'usen per a la comprensió de text de llenguatges naturals, reconeixement d'escriptura a mà, reconeixement de veu, reconeixement de gestos i captura d'imatges.
- GRU: les *Gated Recurrent Unit* contenen dues portes que controlen la manera en que la informació flueix per la unitat. Aquest tipus de xarxes són més senzilles que les *LSTM* degut a que contenen menys parametres, careixen de porta de sortida, són més ràpides i la seva execució és més eficient.

Per tant, les *GRU* tenen millor rendiment en conjunt de dades més petits, en canvi, les *LSTM* al treballar amb conjunt de dades més amplis, poden proporcionar millors resultats davant de situacions més complexes [22].

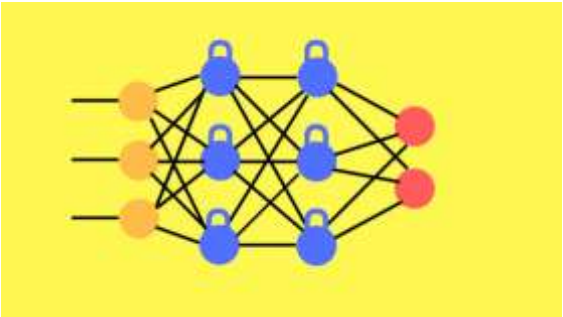

Figure 2.8: Xarxa *LSTM* on podem observar que conté neurones ocultes amb *loops* cap enrere en blau (aquest fer permet que puguin emmagatzemar informació en cel·les de memòria)

2.5.2 Xarxes neuronals convolucionals

Les xarxes neuronals convolucionals (anomenades en anglès Convolutional Neuronal Network, amb els acrònims de CNN o ConvNets) han estat creades per a processar dades que venen en forma de múltiples matrius, per exemple, una imatge en color composta per tres matrius 2D que contenen intensitats de píxels en els tres canals de color. Moltes modalitats de dades tenen la forma de múltiples matrius: les matrius 1D, utilitzades per a senyals i seqüències (inclòs el llenguatge); matrius 2D, usades per a imatges o espectrogrames d'àudio; i les 3D, per a vídeo i imatges volumetriques [18]

Les ConvNets estan formades per neurones que tenen paràmetres en forma de pesos i biaixos que es poden aprendre. Les xarxes neuronals convolucionals fan la suposició explícita del fet que les entrades són imatges, això permet codificar certes propietats en l'arquitectura per reconèixer elements concrets en aquestes. Per fer-nos una idea de com funcionen aquestes xarxes, pensem en com els humans reconeixem les coses. Si veiem una cara, la reconeixem perquè té ulls, nas, boca, orelles, etc. Per tant, per decidir si el que observem és una cara o no, ho fem com si tinguéssim unes caselles mentals de verificació de les característiques que anem marcant. Algun cop, a una cara "li pot faltar" una orella al estar tapada pel cabell, però igualment la classifiquem amb una certa probabilitat que sigui una cara perquè hi veiem els ulls, nas i boca.

Però abans de tot això, les ConvNets han de saber com és una orella o un nas per saber si es troben a la imatge o no; és a dir, prèviament s'ha d'identificar línies, límits, formes o textures que siguin similars a les que contenen les orelles o nassos. Aquesta tasca és la que les xarxes convolucionals tenen encomanat fer. A més a més, aquestes han de poder identificar com es troben localitzades les parts d'una cara entre si, ja que si això no fos així, la cara no s'assemblaria al concepte que estem acostumats[23].

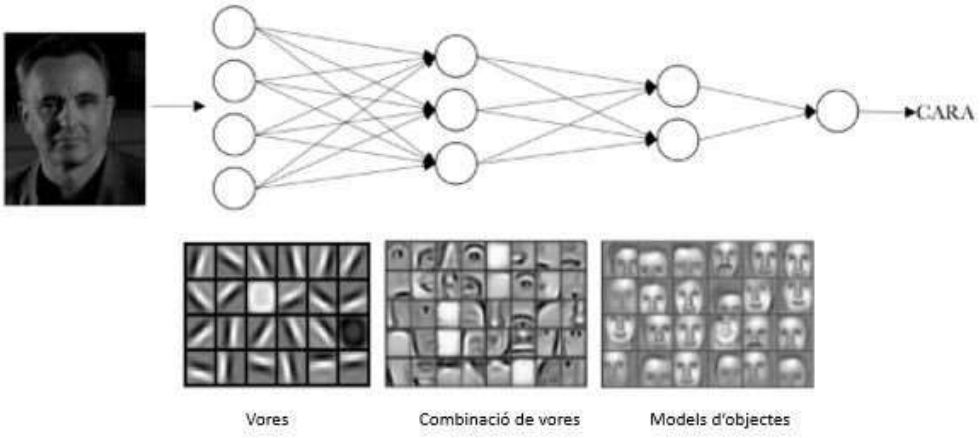

Figure 2.9: En una ConvNet, cada capa va aprenent diferents nivells d'abstracció. Amb multiples capes és pot conseguir identificar estructures més complexes en les dades d'entrada. Recurs: *DEEP LEARNING INTRODUCCIÓN PRÁCTICA CON KERAS*.

L'èxit recent de les ConvNets és el reconeixement facial. S'ha de destacar que les imatges es poden etiquetar a escala de píxel això tindrà aplicacions en tecnologia, inclòs robots mòbils autònoms, cotxes autònoms o altres aplicacions que inclouen la comprensió del llenguatge natural i reconeixement de veu. Malgrat aquests èxits, les ConvNets es van abandonar fins a la competició ImagNet de l'any 2012. Quan es van aplicar xarxes convolucionals profundes a un conjunt de dades d'aproximadament un milió d'imatges amb 1000 classes diferents van reduir gairebé a la meitat els percentatges d'error. Aquest èxit es deu a l'ús eficient de les GPU's , ReLUs, una nova tècnica de regularització anomenada DroPout62 i tècniques per a generar més exemples d'entrenament deformant els existents.

Molta gent, creu que l'inici del *Deep Learning* va sorgir a partir de l'any 2012 gràcies a aquests avenços, però això no és així, ja que l'*Aprenentatge Profund* va sorgir molts anys enrere, concretament l'any 1955 gràcies a Oliver Selfridge que va ser el que va introduir el concepte d'aprenentatge de patrons. Tan sols el que es va fer a partir del 2012, és aplicar tècniques noves als conceptes bàsics del DL que ja s'havien inventat entre les dècades de 1950 i 1980. [24]

Les arquitectures recents de les ConvNets tenen entre 10 i 20 capes de ReLU, centenars de milions de pesos i milers de milions de connexions entre unitats. Vuit anys enrere, realitzar els entrenaments d'aquestes xarxes tan grans hauria comportat dues setmanes de temps; actualment, gràcies als avenços en la paral · lelització del hardware, software i algoritmes ha reduït aquests temps d'entrenament a tan sols unes poques hores.

Hi ha diversos algoritmes o models de xarxes convolucionals, a continuació, es co-

mentaran els més importants[25]:

- FCN (*Fully Convolutional Network)*: És un algoritme popular per fer segmentacions semàntiques[1]. Aquest model només utilitza diversos blocs de convolució i capes màximes de grup per a descomprimir primer una imatge a 1/32 de la seva mida original. Seguidament, fa una predicció de classe a aquest nivell de granularitat. Finalment, utilitza capes de mostreig i desconvolució per canviar la mida de la imatge a les seves dimensions originals.

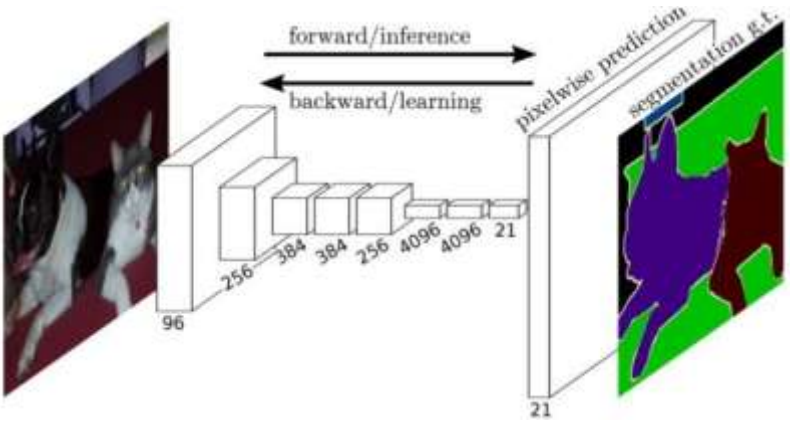

Figure 2.10: Fully Convolutional Network

Aquest tipus de model normalment no tenen capes completament connectades. L'objectiu dels passos de mostreig descendent és captar informació semàntica/contextual, mentre que l'objectiu del mostreig ascendent és recuperar informació espacial. Per recuperar completament la informació espacial de gra fi que es perd en el mostreig descendent s'utilitzen connexions de salt. Les connexions de salt, són unions que omet almenys una capa.

- U-Net: L'arquitectura U-Net es basa completament en la FCN, i es modifica de manera que produeix una millor segmentació en la imatge mèdica, per tant, aquest algoritme s'utilitza per a la segmentació d'imatges biomèdiques. Les principals diferencies respecte a la FCN, és que l'arquitectura U-Net és simètrica i que les connexions de salt entre el camí de mostreig descendent i el camí de mostreig addicional s'aplica un operador de concatenació en comptes d'una suma. Aquestes connexions de salt tenen la intenció de proporcionar informació local a la informació global mentre es realitzen mostres addicionals. A causa de la seva simetria, la xarxa té un gran nombre de mapes de funcions en el camí de mostreig, que permet transferir informació..

[1]La segmentació semàntica és un algoritme de DL que té l'objectiu d'etiquetar cada píxel d'una imatge amb una classe o etiqueta del que s'està representant. [26]

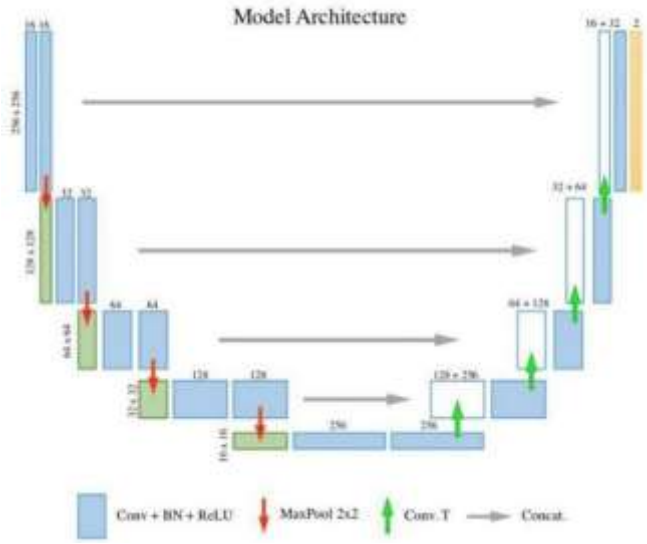

Figure 2.11: U-Net

- R-CNN: L'objectiu de R-CNN és capturar una imatge i identificar correctament on es troben els objectes principals a partir de caixes delimitadores a la imatge. Aquest algoritme crea aquestes caixes mitjançant un procés anomenat cerca selectiva; a grans trets, mira la imatge a través de caixes de diferents mides i per a cada mida intenta agrupar píxels adjacents per textura, color o intensitat per tal d'identificar així objectes.

  Un cop crea les propostes d'objectes, deforma la regió a una mida quadrada estàndard i la transmet a una versió modificada d'AlexNet (un altre algoritme que va inspirar a R-CNN). A la capa final, aquest algoritme amb una màquina vectorial classifica si es tracta d'un objecte i si és així de quin objecte es tracta.[27]

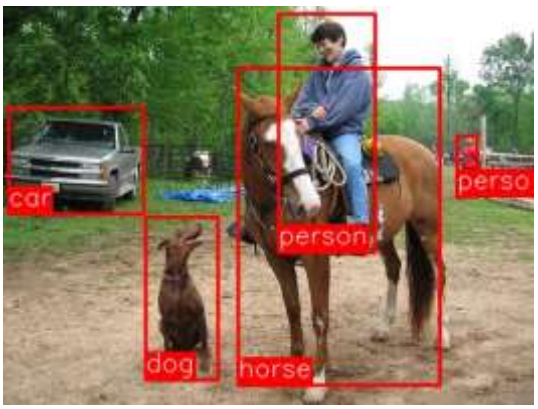

Figure 2.12: RCNN. Recurs: *https://arxiv.org/abs/1311.2524*

- Mask R-CNN: La Mask R-CNN és un model de detecció d'objectes desenvolupat per un grup d'investigadors de Facebook AI el 2017 i és tan sols una evolució de la clàssica R-CNN. Aquest algoritme pot separar diferents objectes, retornant tant la caixa delimitadora com la mascara per a cada objecte que aparegui dins una imatge o d'un vídeo. La màscara R-CNN està composta per *Faster R-CNN* i *FCN*. *Faster R-CNN* és un molt bon algoritme per a la detecció d'objectes. Aquesta, consta de dues etapes que estan connectades mitjançant una columna vertebral[2]:

  - La primera etapa, *RPN* (en català, xarxa de propostes de regió), genera propostes sobre les regions on hi pot haver un objecte a partir de la imatge d'entrada.

  - La segona etapa, que és essencialment *Faster R-CNN*, prediu la classe de l'objecte, refina la caixa delimitadora i genera una màscara al nivell de pixels de l'objecte segons la proposta de la primera etapa.

La màscara *RCNN* és conceptualment senzilla: la *Faster R-CNN* té dues sortides per a cada objecte candidat, una etiqueta de classe i un desplaçament de caixa delimitadora; a això hi hem d'afegir una tercera branca que genera la màscara d'objecte, una màscara binària que indica els píxels on es troba l'objecte al quadre de delimitació. Però la sortida de la màscara addicional és diferent de les sortides de classe i caixa, que requereixen l'extracció d'un disseny espacial molt més fi d'un objecte. Per fer-ho, la Mask R-CNN usa la xarxa *FCN*.[28]

[2]

- Backbone és una xarxa neuronal profunda d'estil FPN (*Feature Piramid Networks*) i és la columna vertebral. Aquesta conté dos camins, un ascendent i un descendent. El camí ascendent que pot ser qualsevol ConvNet, extreu funcions d'imatges en brut. El camí descendent genera un mapa de piràmides de característiques.

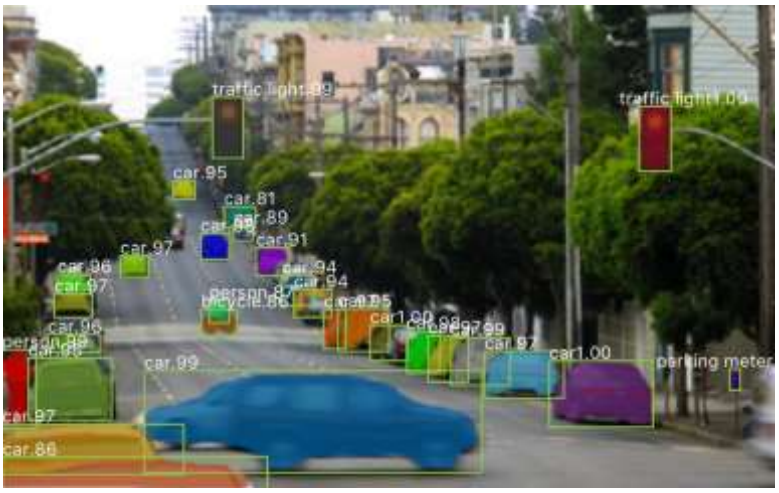

Figure 2.13: Mask RCNN. Recurs: *Mask RCNN paper*

Com a resum de la Mask R-CNN podem dir que aquest és un molt bon algoritme que prediu l'etiqueta de la classe i el percentatge, el quadre de delimitació i la màscara dels objectes d'una imatge o d'un vídeo.

2.6 Eines per a la comunitat de Deep Learning

Anys enrere, el *Deep Learning* requeria experiència en llenguatges com C++ i CUDA; actualment, tenint coneixements bàsics de Python és suficient. Aquest fet ha estat gràcies al gran nombre d'eines de software que han anat apareixent amb el pas dels anys, com Keras o TensorFlow. Aquestes eines anomenades *frameworks*, faciliten enormement el desenvolupament i entrenament dels models i a més a més permeten abstreure les peculiaritats del hardware al desenvolupador de l'algoritme per tal d'accelerar el procés d'entrenament.

Algunes de les eines que més destaquen són TensorFlow, Keras u Pytorch, ja que actualment són dels més dinàmics que hi ha.

TensorFlow és l'eina dominant per excel·lència; va ser desenvolupat per investigadors i enginyers del grup Google Brain. Aquest sistema va ser desenvolupat per facilitar la investigació en *Machine Learning* i així fer més ràpidament la transició d'un prototip d'investigació a un sistema de producció.

A l'anterior *framework* el segueix Keras amb una API d'alt nivell per a xarxes neuronals, això la converteix en l'entorn perfecte per iniciar-se en el tema. El codi s'especifica en Python, i en aquests moments és capaç d'executar-se a sobre de tres entorns destacats com son TensorFlow, CNTK o Theano. Una altra de les característiques és que l'usuari pot canviar el motor d'execució sense haver de canviar el seu codi Keras.

PyTorch i Torch són dos entorns de ML implementats amb llenguatge C, per treure profit d'infraestructures altament paral·leles. PyTorch és la versió més focalitzada per a DL i basada en Python, desenvolupada per Facebook [23].

També hem de parlar d'unes altres eines anomenades Data Science. Gràcies a haver-hi aquestes bases de dades obertes a internet s'ha pogut progressar en l'àmbit del DL, moltes d'aquestes han donat suport al ràpid desenvolupament d'algoritmes d'*intel·ligència artificial*. Alguns exemples de bases de dades són ImagNet, Kaggle i Google Colab.

ImagNet és una base de dades oberta a tothom que conté més de 22.000 classes diferents i més de 14 milions d'imatges etiquetades a mà.

Kaggle és una altra plataforma de Data Science i aquesta és la més gran del món amb més d'un milió d'usuaris; és una plataforma excel·lent perquè estudiants aprenguin i creixin en el camp de Data Science i ML

Google Colab és un servei de computació al núvol, basat en les Notebooks de Jupyter[3] que permet l'ús gratuït de les GPUs i TPUs de Google, amb llibreries com: Pytorch, TensorFlow, Keras, OpenCV. Malgrat tenir algunes limitacions, és

[3]Jupyter Notebook és un entorn informàtic interactiu basat en la web per crear documents de Jupyter.

una eina ideal no només per a practicar i millorar els nostres coneixements en tècniques i eines de Data Science, sinó també pel desenvolupament d'aplicacions (proves pilots) de ML i DL. Amb Google Colab es poden crear quaderns o bé importar aquells que ja tinguem creats amb altres aplicacions, a més a més de compartir-los i exportar-los quan vulguem. Aquest servei de computació ens permet treballar amb tota aquella informació que tinguem continguda al nostre propi Google Drive, unitat d'emmagatzematge global, github i inclòs altres sistemes d'emmagatzematge al núvol.

Aquestes són algunes de les eines més destacades orientades a l'*Aprenentatge Profund*.

3 Implementació de l'algoritme

3.1 Visió general

En aquest capítol s'explica l'algoritme de Mask R-CNN muntat a *Google Colab*. L'algoritme s'ha desenvolupat i muntat en aquest servei al núvol, ja que conté GPUs que acceleren l'execució.

Per a poder executar el codi al *Google Colab*, requereix que totes les imatges preparades han d'estar associades a la unitat de *Google Drive* a través del mateix compte de Gmail. Seguidament es mostra la distribució de la carpeta on s'ubiquen tant les imatges preparades, els scripts i la resta d'arxius corresponents a l'algoritme a través d'un esquema:

- TFG
  - Software: Aquesta carpeta conté tot el relacionat amb els algoritmes.
    - ∗ Fruites: a l'interior d'aquesta carpeta hi haurà tots els codis en Python i els arxius dels pesos.
      - · Script de l'entrenament.
      - · Script de la inferència.
      - · Arxiu dels pesos "*coco*" i pesos entrenats.
  - Data: L'interior d'aquesta carpeta conté les imatges preparades per poder entrenar la xarxa.
    - ∗ Train: Imatges preparades per a realitzar l'entrenament.
    - ∗ Test: Imatges preparades per a realitzar el test o prova.
    - ∗ Validation: Imatges per a realitzar la validació.

Seguidament, el capitol es distribueix en tres apartats. El primer (apartat 3.2) explica amb una breu descripció de tots els paquets necessaris per poder desenvolupar correctament l'algoritme (apartat 3.2). El segon (apartat 3.3) explica el codi de l'algoritme de l'entrenament, el tercer (apartat 3.4) explica el codi de la inferència i finalment el quart (apartat 3.5) explica el codi de la matriu de confusió.

La informació amb el material complet així com els diferents scripts de programació es poden descarregar des de GitHub i utilitzar-los sota llicència per a desenvolupaments de nous projectes a l'àmbit del Smart Agriculture [29].

3.2 Instal·lació del projecte i biblioteques

Per a començar a poder programar l'algoritme, s'ha d'accedir a *Google Colab* i crear un quadern. Seguidament, aquest servei al núvol ha de poder tenir accés a *Google Drive* on hi ha les carpetes que contenen les imatges preparades i la resta d'arxius de l'algoritme.

Per tant, *Google Colab* s'ha de muntar a *Google Drive* a través de la comanda que es mostra a la fig. 3.1.

```
[ ] from google.colab import drive
    drive.mount('/gdrive')
    %cd /gdrive/MyDrive/TFG
```

Figure 3.1: Connectar Colab a Drive

El codi desenvolupat ha estat a partir del projecte de Mask R-CNN 2.1. Per obtenir el projecte al quadern és necessari clonar-lo de la web de Mask R-CNN (`https://github.com/matterport/Mask_RCNN`), cercar l'URL per a descarregar el codi i seguidament tornar al quadern i a partir de l'URL instal·lar totes les biblioteques necessàries que marca l'arxiu *"-r requirements.txt"* i a més a més instal·lar el projecte amb la comanda *"!python3 setup.py install"*.

```
[ ] !git clone https://github.com/matterport/Mask_RCNN
    %cd Mask_RCNN
    !pip3 install -r requirements.txt
    !python3 setup.py install
```

Figure 3.2: Instal·lació paquets Mask R-CNN

Un cop instal·lades les biblioteques i el projecte, s'ha de procedir a fer un canvi de versió d'una biblioteca, concretament l'*h5py*. Cal desinstal·lar la versió actual i instal·lar la versió 2.10.

```
[ ] !pip uninstall h5py
    !pip install h5py==2.10.0
```

Figure 3.3: Instal·lació h5py

Perquè pugui executar-se el projecte és necessari que la versió de TensorFlow sigui superior o igual a la versió 1.3. La comanda *"%tensor flow_version 1.x"* instal·la la versió més actualitzada de la versió 1. Finalment cal descarregar l'arxiu que

conté els pesos anomenats *"cocos"*, aquest el trobem dins el següent web `https://github.com/cocodataset/cocoapi.git`; per obtenir els pesos s'ha de realitzar la mateixa acció que s'ha fet anteriorment amb la Mask R-CNN.

```
[ ] %tensorflow_version 1.x
```

```
[ ] !git clone https://github.com/cocodataset/cocoapi.git
    %cd cocoapi/PythonAPI
    !make
    %cd ../../
```

Figure 3.4: Instalació versió 1.x de TensorFlow i els *cocos*

3.3 Realització i execució de l'algoritme de l'entrenament

En aquest apartat s'explica l'algoritme de l'entrenament i a més a més es segueix mostrant com s'estructura a l'interior de Google Colab (el següent codi ha d'estar justament redactat seguidament de l'apartat 3.2). A les següents imatges es pot observar que a més a més del codi s'hi mostren algunes anotacions, això ho podem observar quan alguna frase comença amb el símbol "#", que especifiquen que s'està fent en cada moment; malgrat que aquestes especificacions seguidament s'explicarà el codi perquè resulti més entenedor. L'algoritme de l'entrenament s'ha dividit en sis parts:

- **1- Importacions de paquets i directoris**:

A la imatge 3.5 s'observa tot un seguit d'importacions de llibreries i fitxers que seran els responsables que l'algoritme desenvolupat pugui funcionar correctament. Entremig de les importacions de les llibreries i dels fitxers s'incorpora una classe anomenada *ROOT_DIR*, aquesta és el directori on es guarden tots els codis (tal com s'ha explicat al principi del capítol (3.1) aquests és trobaran a l'interior de la carpeta TFG/software/Fruites).

```
[ ] import os
    import sys
    import random
    import math
    import re
    import time
    import numpy as np
    import cv2
    import matplotlib
    import matplotlib.pyplot as plt
    import json
    import imgaug

    # Directori del script
    ROOT_DIR = '/content/drive/My Drive/TFG/software/Fruites'
    # Import Mask RCNN ( model.py, utils.py, config.py -> these files contain the main Mask RCNN implementation.)
    #os.path es una llist i per tant podem fer .append
    #os.path crea una llista de tots els directoris on que python buscara un model que nosaltres demanem.
    #Aqui estem afegint a la llista el nostre directori arrel ROOT_DIR ala( segur que busca dins)
    sys.path.append(ROOT_DIR)  # Per trobar la versió local de la biblioteca
    #de la carpeta mrcn importa els seguents fitxers:
    from mrcnn.config import Config
    #Importa el fitxer utils que té moltes funcions
    from mrcnn import utils
    #Importa el fitxer model que te moltes funcions i classes definides
    import mrcnn.model as modellib
    #Importa el fitxer visualize que te moltes funcions sobre la visualització de les imatges
    from mrcnn import visualize
    #Importa la funció log -->def log(text,array=None) Imprimeix un missatge de text. I, opcionalment, si es proporciona una matriu Numpy, imprimeix els valors de forma, mínim i màxim.
    from mrcnn.model import log
```

Figure 3.5: Importacions de les llibreries i fitxers

El *MODEL_DIR* és el directori on es guardarà la carpeta *logs*, és a dir, *logs* és una carpeta on es guardaran tots els pesos de la xarxa neuronal entrenada, aquesta carpeta es guardarà dins la carpeta TFG/software/Fruites. A *COCO_MODEL_PATH* es guarda la direcció on es troben els pesos inicials, si en aquesta direcció no es troba l'arxiu dels pesos inicials, el descarregarà i es guardarà en aquesta direcció.

```
%matplotlib inline
# Directori per guardar els logs i models de punt de control, si no es proporciona
# mitjançant la comanda line d'ordres --logs
MODEL_DIR = os.path.join(ROOT_DIR, "logs")

# Direcció local on estan el fitxer dels pesos
COCO_MODEL_PATH = os.path.join(ROOT_DIR, "logs/mask_rcnn_coco.h5")
# Mira si existeix el fitxer mask_rcnn_coco.h5 i si no hi és el descarrega
if not os.path.exists(COCO_MODEL_PATH):
    %cd software/Fruites/logs
    utils.download_trained_weights(os.path.join(os.path.abspath("../"),"mask_rcnn_coco.h5"))
    %cd ../../../../
```

Figure 3.6: Directori on es guarda la xarxa sense entrenar

Seguidament a la imatge 3.7 s'introdueixen tres classes: *DATASET_TRAIN_DIR*, *DATASET_VAL_DIR i DATASET_TEST_DIR;* en aquestes es guarda la ruta on es troben les carpetes de les imatges d'entrenament, prova i validació. Les accions de *"from tensorflow.compat.v1 import...."* són importacions de llibreries que configuren el mode de treball de les llibreries de TensorFlow. La comanda de la línia 6,és una classe que té uns valors predeterminats per als paràmetres; la següent acció és una funció que permet créixer a la xarxa neuronal i és un mètode de la classe *config*. Finalment la darrera comanda és una clase per executar TensorFlow amb les configuracions de la classe config.

```
DATASET_TRAIN_DIR = '/content/drive/My Drive/TFG/data/train' #content drive
DATASET_VAL_DIR = '/content/drive/My Drive/TFG/data/validation'
DATASET_TEST_DIR = '/content/drive/My Drive/TFG/data/test'
```

```
from tensorflow.compat.v1 import ConfigProto
from tensorflow.compat.v1 import InteractiveSession
```

```
config = ConfigProto()
config.gpu_options.allow_growth = True
session = InteractiveSession(config=config)
```

Figure 3.7: Direccions de directoris, importacions de llibreries i altres funcions

- **2- Configuracions**:

A la imatge 3.8 es troba el codi de la configuració del conjunt de dataset Fruites de l'entrenament. Aquesta, deriva de la classe *Config* i substitueix alguns valors. Per a construir el model Mask R-CNN cal especificar diversos paràmetres, ja que aquests controlen la supressió no màxima (NMS), la intersecció per unió (IoU), la mida de la imatge, el nombre de ROI per imatge, la capa d'agrupació de ROI, entre d'altres.

Com s'acaba d'explicar al punt 1, la classe *config* té uns valors predeterminats per als paràmetres, per a poder canviar alguns valors d'aquests paràmetres es crea la classe *FruitesConfig*.

S'assigna a *NAME* un nom reconeixible per a la configuració. Un dels paràmetres crítics és el nombre de classes (*NUM_CLASSES)*, que determina totes les classes diferents que té l'algoritme; el fons o background s'ha de considerar com una classe, és a dir, per aquest algoritme es tenen 3 classes (pomes, peres i el background).

Les passades per època (*STEPS_PER_EPOCH)* és un altre paràmetre crític que s'ha de configurar. Aquest paràmetre determina el nombre de passades que es fan a cada entrenament; aquest valor ha estat determinat que sigui igual al nombre d'imatges d'entrenament, és a dir, 153.

Finalment, *DETECTION_MIN_CONFIDENCE* és una variable que permet quan s'agafa una imatge de la carpeta validació durant l'entrenament o l'actualització de pesos i calcula l'error, si aquest és inferior a 0,8 adquireix la configuració de pesos anteriors; per tant, si l'error és bo actualitzarà els pesos. Amb aquest mecanisme es garanteix l'aprenentatge de l'algoritme.

- **3- Conjunt de dades:**

Abans de començar el conjunt de dades hem d'inserir les dues següents línies de codi. La funció que fan aquestes és inserir text en imatges.

La classe FruitesDataset forma un conjunt de dades que contenen l'etiquetatge "*Dataset",* aquesta classe simplement emmagatzema la informació sobre totes les imatges d'entrenament a les llistes.

```
class FruitesConfig(Config):
    """Configuration for training on the fruites  dataset.
    Derives from the Base Config class and overrides some values.
    """
    # Nom reconeixible per a la configuració
    NAME = "Fruites"

    # Utilitzee una GPU amb memòria d'11 GB, que s'adapta a UNA imatge.
    # Si s'utilitza una més petita cal ajustar-la.
    IMAGES_PER_GPU = 1

    # Nombre de classes
    NUM_CLASSES = 1 + 1 + 1  # Background + Pomes + Peres

    # Nombre de passades d'entrenament per epoca
    STEPS_PER_EPOCH = 155
    # Iepoca=calcular el gradient per totes les imatges i amb això sumar a cada pes el learning rate*gradient
    # es el nombre d'imatges que es van a resoldre i van a formar epoca
    #learning rate només hi ha 1, gradients tants com pesos
    #no s'ha de tocar, son imatges

    # Omet les deteccions amb <90% de confiança
    DETECTION_MIN_CONFIDENCE = 0.9
    #aquesta variable permet que si l'entrenament, o l'actualitzacio de pesos, quan el programa agafi una imatge de la carpeta val
    #i calculi l'error, si aquest es inferior a 0.8 agafa la configuracio de pesos anteriors, si es bo, actualitza els pesos. Aquest mecanisme garanteix l'aprenentatge

config = FruitesConfig()
config.display()
```

Figure 3.8: Configuració per a l'entrenament del conjunt de dades Fruites

```
from PIL import Image,ImageDraw
```

```
axis_Width = 15
```

Figure 3.9: Funcions per afegir text en imatges

La primera línia del mètode *load_dataset* crida al mètode *Self.add_class* per definir dues classes. La primera anomenada poma amb el valor de l'identificador igual a 1 i la segona anomenada pera amb el valor de l'identificador igual a 2. A més a més, hi ha una altra classe amb l'identificador de valor 0, aquesta és el fons o background. El background no l'hem d'afegir explícitament, ja que existeix per defecte. L'última línia del *load_dataset* crida al mètode *add_image* per afegir una imatge al conjunt d'imatges, aquesta acció tan sols es produeix si existeix la imatge i alhora el fitxes que s'anomena igual que la imatge, però que acaba en *.json*.

El *extract_masks* és un mètode que extreu per a cada imatge totes les màscares que hi ha. Per a realitzar aquesta acció, l'algoritme necessita el nom del fitxer on es troba la imatge (*filename)* i el camí on es troba la imatge. El bucle *for* dins d'aquest mètode, la funció que fa és contar quants contorns hi ha en cada imatge, és a dir, quants contorns s'anomenen poma o pera. Un cop contades, es genera una matriu buida 3x3 que conté tots els punts on se suposa que es troben les fruites (alçada i amplada en píxels de contorn, i en profunditat el nombre de contorns). Seguidament es torna a repetir el bucle, però aquest cop guardant els valors a la matriu buida generada. Seguidament amb *load_mask* es carreguen les màscares dels conjunts de dades de pomes i peres.

```
class FruitesDataset(utils.Dataset):
    def load_dataset(self, dataset_dir):
        self.add_class('dataset', 1, 'poma')
        self.add_class('dataset', 2, 'pera')
        # find all images
        for i, filename in enumerate(os.listdir(dataset_dir)):
            annotation_file = os.path.join(dataset_dir, filename.replace('.jpg', '.json'))
            if '.jpg' in filename and os.path.isfile(annotation_file):
                    self.add_image('dataset',
                                   image_id=i,
                                   path=os.path.join(dataset_dir, filename),
                                   annotation=annotation_file)

    def extract_masks(self, filename):
        json_file = os.path.join(filename)
        with open(json_file) as f:
            img_anns = json.load(f)

        n_masks = 0
        for anno in img_anns['shapes']:
            if anno['label']=='poma' or anno['label']=='pera':
                n_masks+=1

        masks = np.zeros([img_anns['imageHeight'], img_anns['imageWidth'], n_masks], dtype='uint8')
        classes = []
        i=0
        for anno in img_anns['shapes']:
            if anno['label']=='poma' or anno['label']=='pera':
                mask = np.zeros([img_anns['imageHeight'], img_anns['imageWidth']], dtype=np.uint8)
                cv2.fillPoly(mask, np.array([anno['points']], dtype=np.int32), 1)
                masks[:, :, i] = mask
                classes.append(self.class_names.index(anno['label']))
                i+=1
        return masks, classes

    # load the masks for an image
    def load_mask(self, image_id):
        # get details of image
        info = self.image_info[image_id]
        # define box file location
        path = info['annotation']
        # load masks
        masks, classes = self.extract_masks(path)
        return masks, np.asarray(classes, dtype='int32')

    def image_reference(self, image_id):
        info = self.image_info[image_id]
        return info['path']
```

Figure 3.10: Classe FruitesDataset

Abans d'executar el codi de la fig. 3.11, s'ha d'executar el codi de la fig. 3.12. Si aquest no es pot executar, s'ha de fer córrer el següent codi. El següent codi tan sols és una altra manera de muntar el *Google Colab* al *Google Drive*, és a dir, és una solució al problema.

```
#Executar 2n
from google.colab import drive
drive.mount('/content/drive')
```

Figure 3.11: Un altre metode de muntar el Colab al Drive

Seguidament es prepara el conjunt de dades per a cada conjunt d'imatges per al seu ús. És a dir, prepara el *dataset* de l'entrenament, de la prova i de la validació. Simplement crea més atributs sobre les dades, com ara el nombre de classes o el nombre d'imatges, entre d'altres.

El següent codi serveix per a carregar i mostrar cinc imatges aleatòries de les imatges que tenim al *dataset_train* amb els contorns dels productes marcats.

```
# Create data sets
# train set
dataset_train = FruitesDataset()
dataset_train.load_dataset(DATASET_TRAIN_DIR)
dataset_train.prepare()
print('Train: %d images' % len(dataset_train.image_ids))

# val set
dataset_val = FruitesDataset()
dataset_val.load_dataset(DATASET_VAL_DIR)
dataset_val.prepare()
print('Val: %d images' % len(dataset_val.image_ids))

# test set
dataset_test = FruitesDataset()
dataset_test.load_dataset(DATASET_TEST_DIR)
dataset_test.prepare()
print('Test: %d images' % len(dataset_test.image_ids))
```

```
# Càrrega i mostra imatges aleatoriament


image_ids = np.random.choice(dataset_train.image_ids, 5)
for image_id in image_ids:
    image = dataset_train.load_image(image_id)
    mask, class_ids = dataset_train.load_mask(image_id)
    visualize.display_top_masks(image, mask, class_ids, dataset_train.class_names)
```

Figure 3.12: Preparació del dataset i acció de càrregar i mostrar imatges aleatories

Finalment abans de crear el model és càrrega l'extensió de TensorBoard, aquesta és una eina de gràfics que permet examinar el model de TensorFlow.

```
# Load the TensorBoard notebook extension
%load_ext tensorboard
```

Figure 3.13: Extensió de Tensorboard

- **4- Model**:

Arribats a aquest punt toca definir el model, fins ara els codis escrits només han servit per a definir el conjunt de dades amb les coordenades de cada etiqueta. El següent codi crea una instància de la classe *modellib. MaskRCNN*, que construeix l'arquitectura del model Mask R-CNN. El paràmetre *mode* està configurat en entrenament per indicar que es formarà el model.

Un cop creada l'arquitectura del model, els pesos es carreguen mitjançant el mètode *model.load_weights*. Aquest mètode conté tres paràmetres. El primer és l'adreça del fitxer de pesos. El segon serveix per decidir si s'assigna el pes de les capes segons els seus noms; finalment, l'exclude són el nom de les capes per a les quals no carreguem els seus pesos (aquestes capes són les capdavanteres de l'arquitectura).

Quan es du a terme l'entrenament per primer cop, s'ha d'iniciar amb els pesos coco, ja que inicialment no es disposa d'altres pesos que no siguin aquests, per aquest motiu *init_with= "coco"*; un cop tenim algun pes entrenat, ja podem canviar *coco* per *last*; d'aquesta manera l'entrenament agafa els valors dels darrers pesos entrenats.

La darrera acció (%tensorboard) serveix per a crear la gràfica que mostra l'índex d'error dels pesos durant l'entrenament.

```
# Crea el model en mode entrenament
model = modellib.MaskRCNN(mode="training", config=config, model_dir=MODEL_DIR)
```

```
print(model.find_last())
```

```
init_with = "last"  # imagenet, coco, or last(siusem)  --> despres de  fer el 1r entrennamennt podem ferho amb last

if init_with == "imagenet":
    model.load_weights(model.get_imagenet_weights(), by_name=True)
elif init_with == "coco":
    # Carrega pesos entrenats a MS COCO, però ometeu capes que
    # són diferents a causa del diferent nombre de classes
       model.load_weights(COCO_MODEL_PATH, by_name=True,
                          exclude=["mrcnn_class_logits", "mrcnn_bbox_fc",
                                   "mrcnn_bbox", "mrcnn_mask"])
elif init_with == "last":
    # Carrega l'últim model entrenat i continua amb l'entrenament.
    model.load_weights(model.find_last(), by_name=True)
```

```
%tensorboard --logdir /content/drive/MyDrive/TFG/software/Fruites/logs
```

Figure 3.14: Codi del model

- **5. Entrenament**:

La fase d'entrenament, aquí s'entrena la xarxa i s'obtenen uns valors dels pesos determinats. Per dur a terme aquesta fase, es crea una classe anomenada *fruita_augmentation*, el que fa aquesta classe és amb una probabilitat del 50% agafa les imatges que es tenen i les gira un cert angle aleatori (entre 360 graus i -360 graus) per així generar noves imatges. L'objectiu d'aquesta classe és augmentar el nombre de dades per a realitzar l'entrenament, així mentre més dades tingui la xarxa millors resultats proporcionarà.

Al model d'entrenament es fan còrrer tant les imatges d'entrenament com les de prova. Les imatges de prova serveixen per fer la comprovació amb una imatge al final de cada època i així mesurar el learning_rate (tassa d'aprenentatge). Cada

època conté les 153 passades que hem definit anteriorment a la configuració. Amb aquest algoritme s'entrenarà 100 èpoques i només amb les capes de les neurones del final. Les primeres capes de les neurones, ja estan pre-entrenades.

Un cop acabat aquest entrenament, es realitza una fase d'afinació que serveix per millorar encara més la precisió de l'algoritme. Aquesta fase s'ha realitzat amb un *learning rate* deu vegades més petit i amb 24 èpoques més que anteriorment.

```
# Entrenament


fruita_augmentation = imgaug.augmenters.Sometimes(0.5,
    [imgaug.augmenters.geometric.Affine(rotate=(-360,360))])


model.train(dataset_train, dataset_test,
            learning_rate=config.LEARNING_RATE,
            epochs=100,
            layers='heads',
            augmentation = fruita_augmentation )
```

```
model.train(dataset_train, dataset_test,
            learning_rate=config.LEARNING_RATE / 10,
            epochs=124,
            layers="all")
```

Figure 3.15: Codi d'entrenament

3.4 Realització i execució de la inferència

En aquest apartat s'explica l'algoritme de la inferència i igual que l'apartat anterior se segueix mostrant com s'estructura a l'interior de Google Colab. La inferència és el pas on s'ha de posar en pràctica tot allò que l'algoritme ha après durant l'entrenament per a poder detectar els productes agrícoles en una imatge.

Per a l'execució de la inferència igual que a l'apartat anterior, primer de tot s'ha de realitzar els passos de l'apartat 3.2. A més a més, igual que a l'apartat 3.3 s'han de fer les mateixes importacions de les biblioteques i directoris, també importar les configuracions i el conjunt de dades (excepte la darrera funció de muntar al /content/drive).

Per a la inferència s'agafen els pesos entrenats a l'última època, és a dir, a la 100. També, es mesura el temps que triga a executar-se i es guarda el temps d'inici. *Results* el que fa és realitzar la inferència i aquesta és guardada a results; aquesta classe és una matriu on *rois* és la caixa o marc de l'objecte, *masks* són els punts que resegueixen el contorn de l'objecte, *class_ids* és l'etiqueta de cada objecte i *scores* és la probabilitat que aquest objecte sigui el que diu la classe.

```
class InferenceConfig(FruitesConfig):
    """Configuracio per entrenar kes Fruites dataset
    Deriva de la classe base Config i substitueix alguns valors."""

    GPU_COUNT = 1
    IMAGES_PER_GPU =1
inference_config = InferenceConfig()
```

```
# Recreate the model in inference mode
model = modellib.MaskRCNN(mode="inference",config=inference_config,model_dir=MODEL_DIR)
#model_path = model.find_last()
model_path = '/gdrive/MyDrive/TFG/software/Fruites/logs/fruites20210609T1538/mask_rcnn_fruites_0100.h5'
print("Loading weights from ", model_path)
model.load_weights(model_path, by_name=True)
```

```
# Infer
start_time = time.time()

results = model.detect([image], verbose=1)
print("--- %s seconds ---" % (time.time() - start_time))
r = results[0]

visualize.display_instances(image, r['rois'], r['masks'], r['class_ids'],
                            dataset_test.class_names, r['scores'], show_mask=False, show_bbox=True)
```

Figure 3.16: Codi inferència

Finalment per a la visualització dels resultats s'ha d'inserir el codi de la imatge 3.17. En aquest codi, l'algoritme captura una imatge aleatòriament i és mostra per pantalla les etiquetes que s'han posat a cadascun dels productes agrícoles. A continuació, l'algoritme agafa aquesta mateixa imatge sense les etiquetes i crea les caixes, els contorns, les etiquetes i les probabilitats que aquell producte sigui el que diu l'etiqueta.

```
def get_ax(rows=1, cols=1, size=8):
    """Torna a Matplotlib Axes una matriu on s'usaran
    totes les visualitzacions al quadern.

    Change the default size attribute to control the size
    of rendered images
    """
    _, ax = plt.subplots(rows, cols, figsize=(size*cols, size*rows))
    return ax
```

```
# Test on a random image

image_id = random.choice(dataset_test.image_ids)
original_image, image_meta, gt_class_id, gt_bbox, gt_mask =\
    modellib.load_image_gt(dataset_test, config,
                           image_id, use_mini_mask=False)
log ("original_image", original_image)
log ("image_meta", image_meta)
log ("gt_class_id", gt_class_id)
log ("gt_bbox", gt_bbox)
log ("gt_mask", gt_mask)

visualize.display_instances(original_image, gt_bbox, gt_mask, gt_class_id,
                            dataset_test.class_names, figsize=(8, 8))
```

```
results = model.detect([original_image], verbose=1)

r = results[0]
visualize.display_instances(original_image, r['rois'], r['masks'], r['class_ids'],
                            dataset_val.class_names, r['scores'], ax=get_ax())
```

Figure 3.17: Obtenció de resultats

3.5 Matriu de confusió

L'algoritme de la matriu de confusió permet analitzar com es desenvolupa l'algoritme. Aquest codi no ha estat desenvolupat per l'autor del projecte sinó que s'ha extret del següent link: https://github.com/Altimis/Confusion-matrix-for-Mask-R-CNN del portal *GITHUB*. A la fig. 3.18és mostra la primera part del codi de la matriu de confusió. En aquesta primera part del codi l'algoritme fa els càlculs necessaris per extreure les dades per poder crear la matriu de confusió, és a dir, extreu dades de les imatges predites i de les imatges reals i les agrupa en vectors.

```
import pandas as pd

#ground-truth and predictions lists
gt_tot = np.array([])
pred_tot = np.array([])
#mAP list
mAP_ = []

#compute gt_tot, pred_tot and mAP for each image in the test dataset
for image_id in dataset_test.image_ids:
    original_image, image_meta, gt_class_id, gt_bbox, gt_mask =\
        modellib.load_image_gt(dataset_test, inference_config, image_id, use_mini_mask=False)
    info = dataset_test.image_info[image_id]

    # Run the model
    results = model.detect([original_image], verbose=1)
    r = results[0]

    #compute gt_tot and pred_tot
    gt, pred = utils.gt_pred_lists(gt_class_id, gt_bbox, r['class_ids'], r['rois'])
    gt_tot = np.append(gt_tot, gt)
    pred_tot = np.append(pred_tot, pred)

    #precision_, recall_, AP_
    AP_, precision_, recall_, overlap_ = utils.compute_ap(gt_bbox, gt_class_id, gt_mask,
                                        r['rois'], r['class_ids'], r['scores'], r['masks'])
    #check if the vectors len are equal
    print("the actual len of the gt vect is : ", len(gt_tot))
    print("the actual len of the pred vect is : ", len(pred_tot))

    mAP_.append(AP_)
    print("Average precision of this image : ",AP_)
    print("The actual mean average precision for the whole images (matterport methode) ", sum(mAP_)/len(mAP_))
```

Figure 3.18: Codi Matriu de confusió

Finalment en la darrera part del codi (fig. 3.19), es guarden els vectors amb les dades de les imatges predites i les imatges reals i finalment es mostra la matriu de confusió del model.

```
gt_tot=gt_tot.astype(int)
pred_tot=pred_tot.astype(int)
#save the vectors of gt and pred
save_dir = "output"
gt_pred_tot_json = {"gt_tot" : gt_tot, "pred_tot" : pred_tot}
df = pd.DataFrame(gt_pred_tot_json)
if not os.path.exists(save_dir):
    os.makedirs(save_dir)
df.to_json(os.path.join(save_dir,"gt_pred_test.json"))
```

```
print("ground truth list : ",gt_tot)
print("predicted list : ",pred_tot)

# en el output de esta función se puede ver la clase que corresponde a cada clasificacion
# pues el 0 es el background, el 2 es maduro y el 3 es podrido (comprobando la posicion del vector con la imagen)
# entonces la clase 1 es verde
# CLASE A = BACKGROUND
# CLASE B = poma
# CLASE C = pera




tp,fp,fn=utils.plot_confusion_matrix_from_data(gt_tot,pred_tot,fz=18, figsize=(9,9), lw=0.5)
```

Figure 3.19: Codi Matriu de confusió

4 Resultats

4.1 Visió general

Per validar la implementació de l'algoritme, aquest capítol mostra la capacitat que té l'algoritme per a detectar i classificar a través d'imatges pomes i peres.

En primer lloc, es començarà analitzant l'índex d'error en les époques d'entrenament (veure 4.2) , seguidament s'analitzarà la matriu de confusió (veure 4.3) i finalment el pressupost del projecte (veure 4.4).

4.2 Índex d'error durant l'entrenament

L'entrenament de la xarxa neuronal té una durada d'aproximadament 8 hores, aquest es pot dividir en dues parts. La primera consisteix en un entrenament de les branques capdevanteres (head branches) de 100 èpoques, la segona consisteix en una afinació final (fine-tunning) formada per 24 èpoques més.

A partir del gràfic (fig.4.1) de la xarxa neuronal entrenada amb 124 èpoques, és pot observar com la xarxa neuronal va aprenent amb el pas del temps. A l'inici aquesta falla molt, però de mica en mica va aprenent. A l'època 55 ja havia millorat les pèrdues per sota del 30%. D'aquesta manera, l'error s'ha reduit de manera progressiva fins a arribar a les darreres èpoques on ja s'ha mantingut força estable, de fet l'error a la darrera època ha estat del 0,08285.

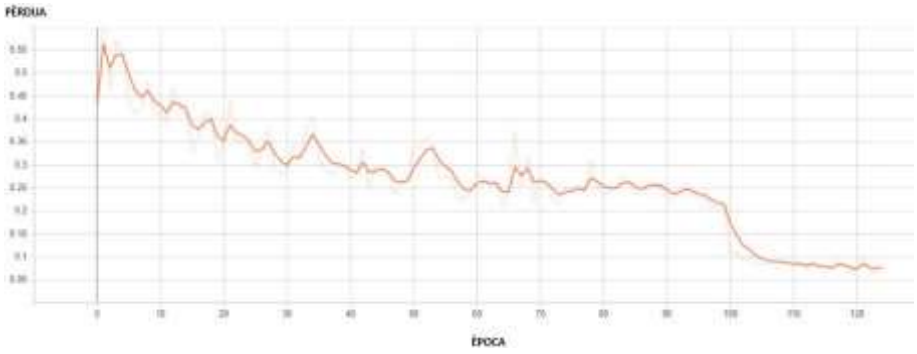

Figure 4.1: Error d'entrenament

4.3 Matriu de resultats

Els resultats de l'algoritme s'obtenen mitjançant la matriu de confusió, una eina que permet visualitzar el funcionament d'aquest. Cada columna de la matriu representa a les instàncies en la classe real, mentre que cada fila de la matriu representa el nombre de prediccions de cada classe. A efectes pràctics aquesta matriu permet veure quin tipus d'encerts i d'errades està tenint el model a l'hora de passar pel procés d'aprenentatge a través de les imatges. Seguidament a la fig. 4.2 és mostra la matriu de confusió obtinguda:

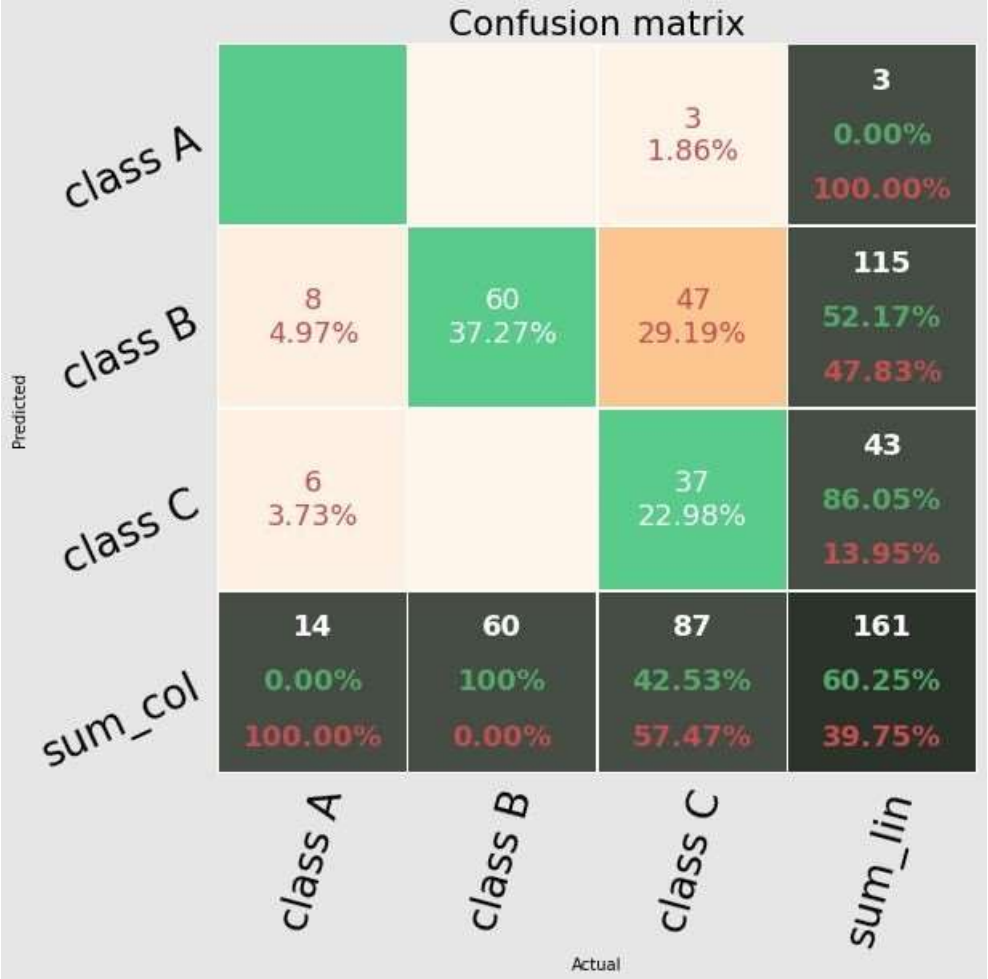

Figure 4.2: Matriu de confusió

La matriu de confusió està formada per tres classes diferents: classe A (pertany al background), classe B (pertany a la poma) i classe C (pertany a la pera).

Les caselles de la matriu de confusió on es creua el mateix tipus de classe (la diagonal amb les caselles verdes) pertany al nombre i percentatge d'encerts que ha tingut

l'algoritme d'aquella classe. En canvi, la resta de caselles són el nombre d'errades que ha tingut l'algoritme.

Per exemple, la cassella de la classe B predita amb la casella de la classe C real obté 47 etiquetes errònies. Això vol dir que l'algoritme classifica erròniament 47 peres com si fossin pomes.

Per tant, la classificació de pomes i peres d'aquest algoritme té una precissió del 60,25 % d'encert. Aquest algoritme classifica les peres amb un 86,05 % d'encert, mentre que les pomes les classifica amb un 52,17 %.

Finalment a les fig. 4.3 , 4.4 i 4.5 es mostren alguns exemples dels resultats que retorna l'algoritme.

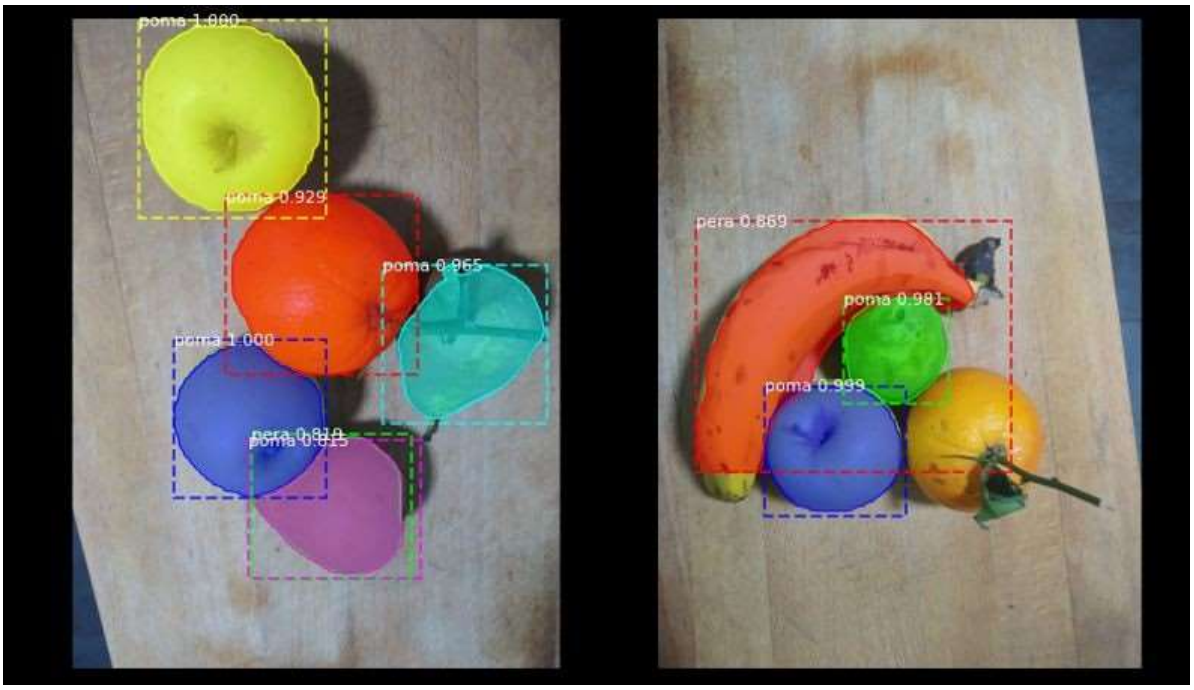

Figure 4.3: Exemples dels resultats obtinguts

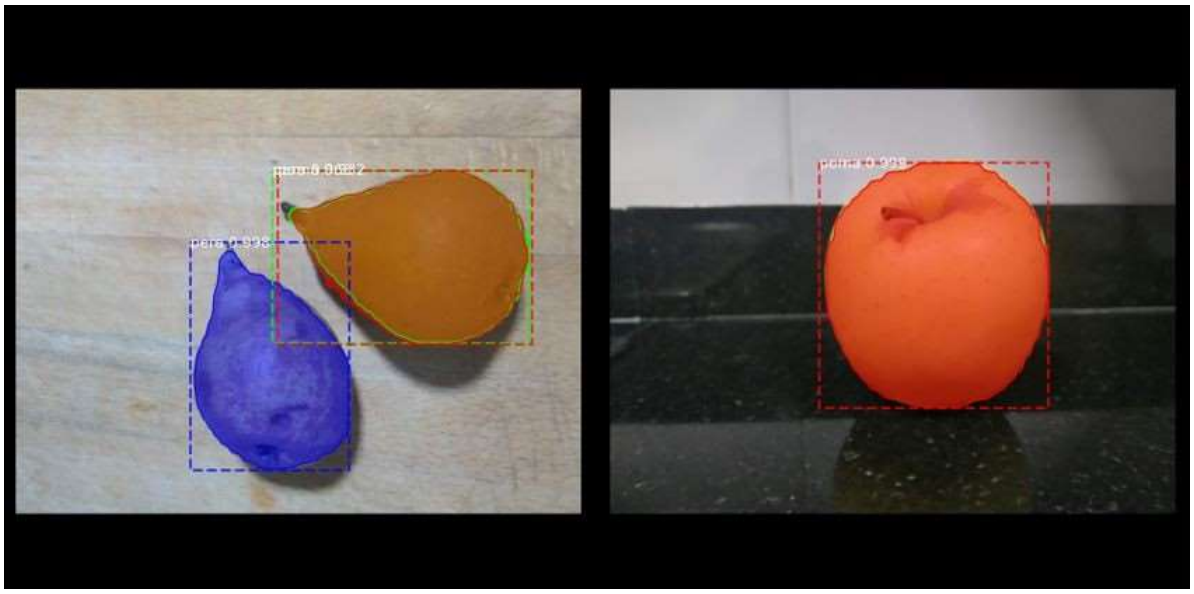

Figure 4.4: Exemples dels resultats obtinguts

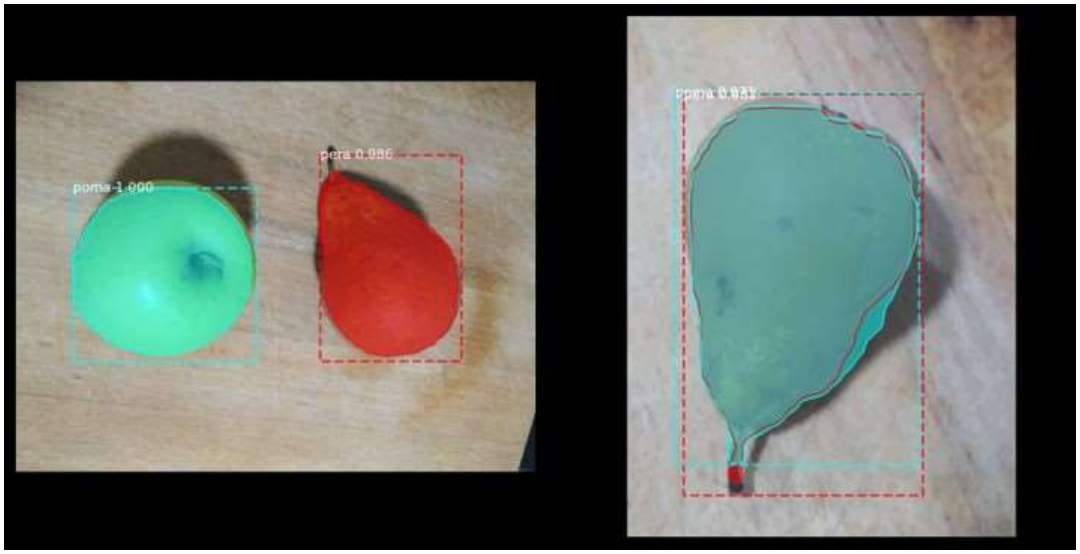

Figure 4.5: Exemples dels resultats obtinguts

4.4 Pressupost

El pressupost de l'algoritme que permet detectar pomes i peres comprèn els costos relacionats amb els costos d'enginyeria i costos variables. Els costos associats al desenvolupament dels programes de software estan inclosos als costos d'enginyeria; donat que els programes utilitzats no tenen cap cost, en el seu lloc es considerarà les

hores de treball dedicades a l'etiquetatge, programació, entrenament, prediccions i avaluació..

4.4.1 Cost d'enginyeria

Per aquest projecte es considera un pressupost total amb una duració de 330 hores repartides segons les activitats de la taula 4.4.1. Per a l'autor es considera una jornada laboral de 17 €/h i de 38 €/h pel director del projecte. El cost total d'enginyeria s'estima en 6030 €.

Activitat	Descripció	Hores	Cost (€)
Obtenció i preparació de dades	Obtenció i etiquetatge de les imatges que l'algoritme de IA ha de processar	20	340
Desenvolupament del model	Programació del algoritme per a detectat productes agricoles	45	765
Entrenament del model	Fase en la que el model utilitza dades d'entrenament per a calcular la predicció.	8	136
Prediccions i avaluació	Fase de realització de prediccions i avalauar els resultats obtinguts	5	85
Gestió del projecte	Redacció de la memòria, reunions entre el director i autor del porjecte, adquisició de coneixements i conceptes de programació, indagació sobre els fonaments del ML amb la finalitat d'adquirir coneixements i aprendre del estat de l'art.	252	4704

Table 4.1: Desglossament dels costos d'enginyeria per a cada activitat

4.4.2 Cost variable

Els costos variables simplement inclouen els costos de la llum, ja que tot aquest projecte ha estat desenvolupat a través d'un ordinador. El consum de l'ordinador és aproximadament 2,2 kWh durant 8 hores de treball.

Tipu de cost	Preu KWH (€)	Hores de treball	KWH consumits	Cost €
Llum	0,20	330	90,2	18

Table 4.2: Cost variable

4.4.3 Cost total

El cost total del projecte és obtingut del total d'hores del desenvolupament del projecte i del cost variable, aquest es resumeix a la taula 4.6.

Pressupost	Cost (€)
Ingenyería	6030
Variable	18
Total	6048

Figure 4.6: Estimació del pressupost total

5 Conclusions i futures liníes de treball

5.1 Conclusions

En aquest projecte, s'ha desenvolupat amb èxit un algoritme de *Machine Learning* basat en aprenentatge supervisat de *Deep Learning* que permet classificar dos tipus de productes agrícoles, pomes i peres. L'algoritme consta d'una xarxa neuronal profunda destinada a resoldre problemes de segmentació semàntica en aplicacions d'aprenentatge automàtic de visió per computadora anomenat Mask R-CNN.

Perquè el model pogués realitzar l'aprenentatge supervisat, aquest ha hagut d'utilitzar imatges que han estat preparades per l'autor a través del programa *Labelme* i extretes tant de bases de dades d'Internet com de càmera fotogràfica. Seguidament, es va procedir a programar i entrenar l'algoritme a través del servei de computació gratuït al núvol anomenat *Google Colab* en codi Python.

Finalment, un cop programat i entrenat l'algoritme, s'han extret els resultats i en aquests s'ha pogut reflectir que l'algoritme ha obtingut un encert de detecció de pomes del 50,17%, de peres del 86,05% i una exactitud general del 60,25%.

Com es pot observar al gràfic de pèrdues (fig. 4.1), aquest té una tendència a disminuir les pèrdues amb el pas del temps, per tant, amb més temps d'entrenament s'obté més precisió. Amb l'entrenament de 100 èpoques, les pèrdues encara no han assolit el seu valor mínim, per tant, es necessita entrenar més èpoques per augmentar l'encert de la màquina.

Analitzant els resultats d'algunes imatges (fig. 4.3, fig. 4.4 i fig. 4.5) es pot observar com encara no hi ha suficients imatges perquè l'algoritme sigui capaç d'obtenir les característiques corresponents per a distingir pomes i peres. Per aquest motiu, per obtenir una precisió més elevada cal entrenar l'algoritme amb un major nombre d'imatges i d'èpoques.

Al finalitzar aquest projecte, es pot afirmar que és possible classificar productes agrícoles a través del processament d'imatges, ja que malgrat tenir poques dades (206 imatges) i faltar entrenament aquest objectiu s'ha pogut assolir.

La informació amb el material complet així com els diferents scripts de programació es poden descarregar des de GitHub i utilitzar-los sota llicència per a desenvolupaments de nous projectes a l'àmbit del Smart Agriculture [29].

5.2 Futures linies de treball

En aquest apartat es presenten noves idees de propostes que es poden desenvolupar després d'aquest projecte. En aquest marc, la preferència hauria de ser realitzar un algoritme que fos capaç de detectar i classificar productes agrícoles amb un alt nivell de precisió, és a dir, que tingués un percentatge d'encert superior al 85%; i que aquest es pogués aplicar en un escenari real d'una aplicació agrícola amb l'objectiu de contribuir en la millora de la qualitat del producte cultivat. Per arribar a assolir el valor de precisió establert cal incrementar molt el nombre dades, és a dir, imatges on apareguin aquests productes. A continuació es mostren un seguit de tasques per poder assolir l'objectiu, les tasques necessàries per assolir aquest objectiu en un futur pròxim s'anomenen CT (que significa curt termini) mentre que les tasques a llarg termini s'anomenen LT.

- Augmentar la precisió (CT): El primer gran objectiu de cara el futur és augmentar la precisió actual de l'algoritme i que aquesta fos superior al 85%. Per poder augmentar-la s'hauria d'obtenir un nombre major de dades preparades.
- Incrementar el nombre de productes (CT): Un altre dels objectius de cara el futur perquè aquest algoritme es pogués aplicar en un escenari real d'una aplicació agrícola seria augmentar el nombre de classes de productes a detectar. És a dir, aquest projecte s'ha limitat al fet que l'algoritme sigui capaç de detectar i classificar dos tipus de classes de fruites (pomes i peres), de cara el futur es pretén que aquest algoritme sigui capaç de detectar gran part de les classes dels productes agrícoles que existeixen arreu del món o si més no els més típics.
- Detectar la maduresa (LT): Aquesta línia de treball consisteix a detectar els diferents graus de maduresa de les classes de productes agrícoles que l'algoritme és capaç de classificar i detectar a través d'imatges.
- Recompte de producte agrícola a través de dron amb càmeres (LT): Una possible aplicació d'aquest algoritme és implementar-lo en un dron perquè mitjançant les càmeres que té incorporades pogués realitzar un recompte ràpid de la quantitat de producte agrícola que hi ha en un camp de conreu.

Agraïments

Vull agrair al director d'aquest projecte, Jose Antonio Soria Pérez per haver-me introduït al món del *Machine Learning* i pels coneixements, temps i consells útils que m'ha proporcionat al llarg de tot el projecte.

Vull agrair tot el suport i paciència a la meva familia, i sobretot als meus pares durant tot els estudis.

Bibliografia

[1] Wikipedia. Revoluciones Industriales. https://es.wikipedia.org/wiki-/Revoluciones_industriales#Primera_revoluci%C3%B3n_industrial

[2] Ayuntamiento de Málaga. Estado del arte y oportunidades de negocio. Technical report.

[3] Alberto De Torres. Inteligencia Artificial: cuál es su estado en 2021 | Sage Advice. https://www.sage.com/es-es/blog/estado-del-arte-de-la-inteligencia-artificial-en-2021/

[4] AgroSpray. Â¿Qué es la agricultura 4.0? Descubre todo al respecto - AgroSpray Blog. https://agrospray.com.ar/blog/agricultura-4-0/

[5] Agrointeligencia. Machine Learning en Agricultura - Agrointeligencia. https://www.agrointeligencia.com/machine-learning-en-agricultura/

[6] ImageNet. ImageNet. https://image-net.org/download.php

[7] Kaggle. Fruits 360 | Kaggle. https://www.kaggle.com/moltean/fruits

[8] COCO. COCO - Common Objects in Context. https://cocodataset.org/-#explore

[9] Wkentaro. GitHub - wkentaro/labelme: Image Polygonal Annotation with Python (polygon, rectangle, circle, line, point and image-level flag annotation). https://github.com/wkentaro/labelme

[10] Matterport Inc. GitHub - matterport/Mask*RCNN* : *MaskR – CNNforobjectdetectionandinstancesegmentationonKerasandT ensorF low. https://github.com/matterport/Mask_RCNN*

[11] Matterport Inc. Matterport, Inc · GitHub. https://github.com/matterport

[12] Google. Te damos la bienvenida a Colaboratory - Colaboratory. https://colab.research.google.com/notebooks/intro.ipynb

[13] BBC. Will a robot take your job? - BBC News. https://www.bbc.com/news/technology-34066941

[14] Oliver Theobald. *Machine Learning For Absolute Beginners*, volume 2. 2017.

[15] IronHack. ¿Qué es Machine Learning? https://www.ironhack.com/es/data-analytics/que-es-machine-learning?utm_source=google&utm_medium=cpc&utm_campaign=120460715959_tck_eur_BCN_data-gen_paid-search_generic_google-ads_dsa_na-site-section_na-ad-size_na-served-type_na-princing&utm_content=search-

[16] BBVA. 'Machine learning': Â¿qué es y cómo funciona? https://www.bbva.com/en/machine-learning-what-is-it-and-how-does-it-work/

[17] SpartanHack. Machine Learning y Deep Learning. Todo lo que necesitas saber - spartanhack. https://spartanhack.com/machine-learning-y-deep-learning-todo-lo-que-necesitas-saber/

[18] Geoffrey LeCun, Yann; Bengio Yoshua; Hinton. Deep learning. Technical report. https://www.cs.toronto.edu/~hinton/absps/NatureDeepReview.pdf

[19] Raymond Perrault, Yoav Shoham, Erik Brynjolfsson, Jack Clark, John Etchemendy, Barbara Grosz Harvard, Terah Lyons, James Manyika, Juan Carlos Niebles, and Saurabh Mishra. 2019 annual report ar intelligence index Project Manager and Report Editor-in-Chief Steering Committee Steering Commitee Introduction Report Highlights Acknowledgements. Technical report.

[20] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2013.

[21] Na8. Breve Historia de las Redes Neuronales Artificiales | Aprende Machine Learning. https://www.aprendemachinelearning.com/breve-historia-de-las-redes-neuronales-artificiales/

[22] Diego Calvo. Red Neuronal Recurrente - RNN - Diego Calvo. https://www.diegocalvo.es/red-neuronal-recurrente/

[23] Jordi Torres. Deep Learning â Introducción práctica con Keras - Jordi TORRES.AI. https://torres.ai/deep-learning-inteligencia-artificial-keras#5_REDES_NEURONALES_CONVOLUCIONALES

[24] Pilar Gómez Gil. El reconocimiento de patrones y su aplicación a las señalesdigitales Editado por. Technical report.

[25] Priya Dwivedi. Segmentació semàntica: arquitectures populars per Priya Dwi- vedi | Cap a la ciència de dades. https://towardsdatascience.com/semantic-segmentation-popular-architectures-dff0a75f39d0

[26] Harshall Lamba. Comprensió de la segmentació semàntica amb UNET | per Harshall Lamba | Cap a la ciència de dades. https://towardsdatascience.com/understanding-semantic-segmentation-with-unet-6be4f42d4b47

[27] Dhruv Parthasarathy. A Brief History of CNNs in Image Segmentation: From R-CNN to Mask R-CNN | by Dhruv Parthasarathy | Athelas. https://blog.athelas.com/a-brief-history-of-cnns-in-image-segmentation-from-r-cnn-to-mask-r-cnn-34ea83205de4

[28] Xiang Zhang. Simple Understanding of Mask RCNN | by Xiang Zhang | Medi- um. https://alittlepain833.medium.com/simple-understanding-of-mask-rcnn-134b5b330e95

[29] Xavier Fresnedoso. xavierf8/FRUITES. https://github.com/xavierf8/FRUITES