UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Barcelona Est

TREBALL FI DE GRAU

Grau en Enginyeria Electrònica Industrial i Automàtica

DETECCIÓ DE MASCARETES EN CARES MITJANÇANT VISIÓ PER COMPUTADOR

Memòria i Annexos

Autor: Ignasi Cervero Corbella
Director: Antoni Grau Saldes

Convocatòria: Juny 2021

Resum

Es presenta el disseny i la implementació d'un sistema de detecció de cares amb i sense mascareta, preparat per poder ser exportat en diferents plataformes. S'ha desenvolupat amb TensorFlow i amb la llibreria OpenCV mitjançant els llenguatges de programació Python i C++.

El model prèviament entrenat SSD MobileNet V2 FPNLite 320x320 ha estat la base de l'entrenament de la xarxa neuronal.

L'entrenament s'ha dut a terme a partir d'una base d'imatges àmplia i diversa, construïda gràcies a la creació d'un procés automatitzat que, donades imatges de cares sense mascareta, es generen cares amb mascareta de diferents tipus, colors i contrasts. A més a més, s'ha augmentat aquesta base d'imatges amb altres processos automatitzats, com el canvi aleatori de la brillantor de la imatge o el redimensionament i col·locació de les cares amb mascareta generades sobre fons aleatoris. Aquestes accions permeten aconseguir més robustesa per a imatges amb múltiples il·luminacions, així com per a cares de diverses mides i col·locades en diferents parts de la imatge.

La situació de la pandèmia de la COVID-19 fa que l'eina pugui tenir diverses aplicacions mentre l'ús de les mascaretes sigui obligatori en espais tancats. Des de control de l'entrada en una porta automàtica a aportar informació a una persona amb visibilitat reduïda.

Paraules clau: *Detecció de mascaretes, TensorFlow, Python, OpenCV, xarxa neuronal SSD.*

Resumen

Se presenta el diseño y la implementación de un sistema de detección de caras con y sin mascarilla, preparado para poder ser exportado en diferentes plataformas. Se ha desarrollado con TensorFlow y con la librería OpenCV mediante los lenguajes de programación Python y C++.

El modelo previamente entrenado SSD MobileNet V2 FPNLite 320x320 ha sido la base del entrenamiento de la red neuronal.

El entrenamiento se ha llevado a cabo a partir de una base de imágenes amplia y diversa, construida gracias a la creación de un proceso automatizado que, dadas imágenes de caras sin mascarilla, se generan caras con mascarilla de diferentes tipos, colores y contrastes. Además, se ha aumentado esta base de imágenes con otros procesos automatizados como el cambio aleatorio del brillo de la imagen o el redimensionamiento y la colocación de las caras con mascarilla generadas sobre fondos aleatorios. Estas acciones permiten conseguir mas robustez para imágenes con múltiples iluminaciones, así como para caras de varios tamaños y colocadas en diferentes partes de la imagen.

La situación de la pandemia de la COVID-19 hace que la herramienta pueda tener varias aplicaciones mientras el uso de las mascarillas sea obligatorio en espacios cerrados. Desde control de la entrada en una puerta automática a aportar información a una persona con visibilidad reducida.

Palabras clave: *Detección de mascarillas, TensorFlow, Python, OpenCV, red neuronal SSD.*

Abstract

The design and implementation of a face wearing and not wearing a face mask detection system are presented, prepared to be exported for different platforms. It has been developed with TensorFlow and with the OpenCV library using the Python and C++ program languages.

The pre-trained model SSD MobileNet V2 FPNLite 320x320 has been the basis of neural network training.

The model has been trained from a wide and diverse dataset, built by the creation of an automated process which, given images of faces not wearing a face mask, generate faces wearing a face mask of different types, colors, and contrasts. In addition, this dataset has been increased with other automated processes, such as random change in image brightness or resizing and placing faces with a mask on random backgrounds. These actions provide more robustness for images with multiple illuminations, as well as for faces of various sizes and placed in different parts of the image.

The situation of the COVID-19 pandemic makes the tool able to have several applications while the use of masks is mandatory in enclosed spaces. From input control on an automatic door to provide information to a person with reduced visibility.

Key words: *Face mask detection, TensorFlow, Python, OpenCV, SSD neural network.*

Índex

RESUM I

RESUMEN II

ABSTRACT III

1. PREFACI 1

1.1. Origen del treball 1

1.2. Motivació 1

2. INTRODUCCIÓ 3

2.1. Objectius del treball 3

2.2. Abast del treball 3

2.3. Metodologia del treball 3

3. CONCEPTES TEÒRICS 5

3.1. Visió per computador 5

3.2. Mètodes de detecció de cares humanes 5

3.2.1. Detector Viola Jones 5

3.2.2. *Deep learning* 8

4. SOFTWARE 14

4.1. C++ 14

4.2. Python 14

4.3. Tensorflow 15

4.4. OpenCV 15

4.5. Dlib 16

4.6. LabelImg 16

4.7. Codi propi 16

5. IMPLEMENTACIÓ 17

5.1. Mètode de detecció 17

5.2. Confecció d'un *dataset* 17

5.2.1. Anotació d'objectes en imatges 19

5.2.2. Generar cares amb mascareta 22

5.2.3. Col·locació aleatòria de cares sobre fons aleatoris 30

5.2.4. Modificació aleatòria de la brillantor de la imatge 35

5.2.5. Conjut de prova i d'entrenament 40
5.2.6. Imatges i anotacions a TFRecords 41
5.2.7. Unió de TFrecords 41
5.3. Configuració prèvia a l'entrenament 41
5.4. Entrenament 42

6. POSSIBLES APLICACIONS 44

7. PROJECTE COM A PUNT DE PARTIDA 45

8. ANÀLISIS DE L'IMPACTE AMBIENTAL 46

CONCLUSIONS 49

ANÀLISI ECONÒMICA 51

BIBLIOGRAFIA 53

ANNEX A: CODIS 56

A1. Afegir text addicional al nom dels fitxers per evitar duplicitats 56
A2. Afegir text addicional al *filename* de les anotacions 57
A3. Canvi aleatori de brillantor en les imatges 58
A4. Concatenar imatges horitzontalment 61
A5. Crear matriu de confusió a partir de model i imatges de *test* 62
A6. Retallar imatges en massa 66
A7. Informació de les extensions trobades dins d'una carpeta 67
A8. Filtrar i esborrar aquelles imatges en les quals no s'ha detectat cares 68
A9. Generar cares amb mascareta 69
A10. Redimensionar imatges en massa 77
A11. Objectes etiquetats, redimensionats i col·locats aleatòriament sobre un fons aleatori 78
A12. Canviar categoria antiga per una nova en l'anotació de les imatges 81
A13. Unió de TFRecords 82
A14. Moure o copiar fitxers en massa. Possibilitat de filtrar per posició i extensió del fitxer 83
A15. Llistar i comptar diferents classes d'objectes en anotació JSON 85
A16. Anotació JSON a XML 86
A17. Detectar cares amb Viola Jones i anotar-les 88
A18. Estructura anotació en XML 91

A19. Anomenar fitxers amb notació incremental i amb possibilitat d'especificar dígits utilitzats i nombre en què començar 91
A20. Provar model amb imatges 92
A21. Provar model per webcam 95
A22. Creació aleatòria de conjunts de *train* i *test* 97
A23. Anotacions XML a TXT per entrenament Viola Jones 100

ANNEX B: RESULTATS 103

B1. Nas i Boca. 24.768 imatges 103
B2. Cara sencera. 24.768 imatges 105
B3. Cara sencera. 344.581 imatges 107
B4. Cara sencera. 4.326 imatges 108

ANNEX C: COMENTARIS SOBRE RESULTATS 111

ANNEX D: RESULTATS VISUALS 112

D1. Deteccions zona boca i nas 112
D2. Deteccions cara sencera 113

1. Prefaci

1.1. Origen del treball

L'any 2020 va irrompre la malaltia de la COVID-19 en forma d'una pandèmia mundial que va afectar tot el món. Aquesta situació va comportar un confinament global durant tres mesos que desembocà en una desescalada prudent i amb l'obligació de garantir la distància entre les persones, la higiene i dur la mascareta posada (1).

La mascareta esdevingué un element clau en la societat, present en qualsevol àmbit públic tant exterior com interior: al carrer, al transport públic, als establiments, a les aules...

A més a més, mentre no arriba la immunitat de grup que ha de proporcionar la vacuna, es potencia l'ús estricte de les mascaretes més segures per compensar l'eliminació d'algunes mesures com per exemple la distància entre persones en concerts a l'interior (2).

1.2. Motivació

A partir de la situació descrita en l'apartat anterior, sorgí un interès especial en desenvolupar una eina actualitzada, original i vistosa per a satisfer una de les necessitats actuals. Combinant aquests interessos amb la visió per computador, una disciplina científica emmarcada en l'extracció d'informació del món físic a partir de l'adquisició i processament d'imatges, es va arribar a un projecte amb l'objectiu principal de detectar mascaretes en cares de persones.

2. Introducció

2.1. Objectius del treball

El principal objectiu del treball és implementar un sistema de detecció de mascaretes en cares que permeti el seu ús en un entorn controlat a temps real. Per a la visió per computador i la classificació en cascada s'utilitzarà la llibreria OpenCV. Per a l'adquisició de models i entrenament de xarxes neuronals s'utilitzarà l'eina Tensorflow.

Més enllà d'assolir un aprenentatge tècnic de totes les eines emprades, també es vol aconseguir un criteri en la presa de decisions per a la maximització de rendiment i minimització d'esforços. El rendiment dependrà de la base d'imatges adquirida o construïda, així com de l'algoritme i la xarxa neuronal emprats per a la detecció.

A més a més, es vol proposar una solució que es pugui executar en els sistemes operatius principals: Linux, macOS i Windows.

2.2. Abast del treball

En aquest projecte el sistema detecta si una cara humana duu mascareta. L'òptim funcionament anirà relacionat amb una bona il·luminació a l'hora de captar les imatges.

2.3. Metodologia del treball

La metodologia que s'ha seguit per a realitzar el treball ha constat d'una etapa important d'aprenentatge de les tecnologies i posada a punt de l'entorn de desenvolupament. Durant aquesta etapa s'han decidit les tecnologies a emprar i el procés a seguir.

A continuació, s'han investigat diferents bases d'imatges i s'han seleccionat alguns *datasets* per a la confecció d'un banc d'imatges final. S'han desenvolupat programes amb el llenguatge de programació Python, que han automatitzat un procés de creació de cares amb mascareta a partir de cares sense mascareta. A més a més, també s'han automatitzat tasques de *data augmentation,* d'anotació d'objectes, de gestió de fitxers, de generació de conjunts de prova i conjunts d'entrenament...

Tot seguit s'ha escollit un model prèviament entrenat com a punt de partida de l'entrenament i s'ha experimentat l'entrenament amb diverses configuracions i bases d'imatges.

A partir de les mateixes imatges s'han realitzat entrenaments amb l'algoritme de detecció Viola Jones i s'ha comparat la seva efectivitat respecte a la del *deep learning*.

Per últim, s'ha realitzat el document de la memòria del Treball de Fi de Grau.

3. Conceptes teòrics

3.1. Visió per computador

La visió per computador és una branca de la ciència que extreu informació del món físic a partir de la captació i del processament de les imatges. Intenta replicar parts de la complexitat de la visió humana i permetre a les màquines identificar i processar objectes en imatges i vídeos de la mateixa manera que ho fan els humans (3). Ha estat una eina molt important en els darrers anys en molts àmbits de l'enginyeria. L'evolució de la visió per computador l'ha fet un aspecte transcendent en l'automatització i desenvolupament de processos industrials, cadenes de producció, robots autònoms...

Alguns dels àmbits en els quals cada vegada és més present aquesta disciplina són (3):

- Cotxes autònoms.
- Detecció i reconeixement facial.
- Realitat augmentada.
- Salut.

3.2. Mètodes de detecció de cares humanes

Una de les grans aplicacions de la visió per computador i objecte d'anys d'investigació, ha estat la detecció de cares humanes. En el marc del projecte, aquesta aplicació ha estat clau per al desenvolupament de diverses investigacions, a més a més, de ser vital per a la creació automàtica d'un *dataset* i per a l'avaluació del software desenvolupat.

3.2.1. Detector Viola Jones

L'any 2001 els investigadors Paula Viola i Michael Jones van presentar un detector de classificadors en cascada anomenat Detector Viola Jones (4). Pioner en el moment del seu llançament, aquest detector és molt famós gràcies al seu baix cost computacional i alta precisió a l'hora de detectar cares.

Aquest detector, altrament conegut com a *Haar-Cascade Classifier*, és un mètode d'aprenentatge automàtic que tot i haver-se presentat amb la intenció de detectar cares, es pot entrenar per a detectar qualsevol mena d'objecte.

3.2.1.1. Haar *features*

Per aconseguir la detecció d'objectes, l'algoritme es basa a identificar les característiques Haar de la imatge i trobar el patró comú d'aquestes en les imatges positives aportades en el *dataset*.

Figura 1. Haar *features* (Font: (5)).

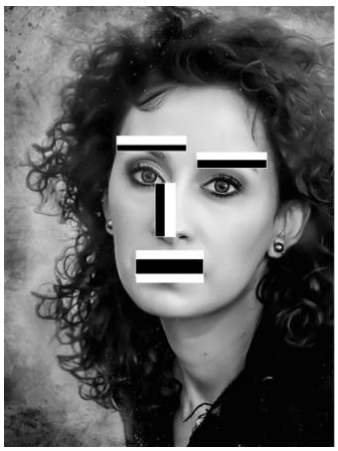

Figura 2. Cara definida per característiques Haar (Font: (6)).

3.2.1.2. Imatge integral

Per optimitzar la detecció de les característiques es converteix la imatge original en una imatge integral. D'aquesta manera, iterant tan sols una vegada per la imatge original s'obté potencialment tota la informació necessària que es necessita per a determinar noves *features.* Cada píxel de la imatge integral conté la suma del valor dels píxels situats a l'àrea que es comprèn a dalt a l'esquerra del mateix píxel i d'ell mateix (7).

$$ii(x,y) = \sum_{x' \leq x, y' \leq y} i(x', y') \qquad \textbf{(Eq. 1)}$$

Essent $ii(x,y)$ la imatge integral i $i(x',y')$ la imatge original, l'(Eq. 1 descriu la creació de la matriu de la imatge integral a partir dels punts de la matriu de la imatge original.

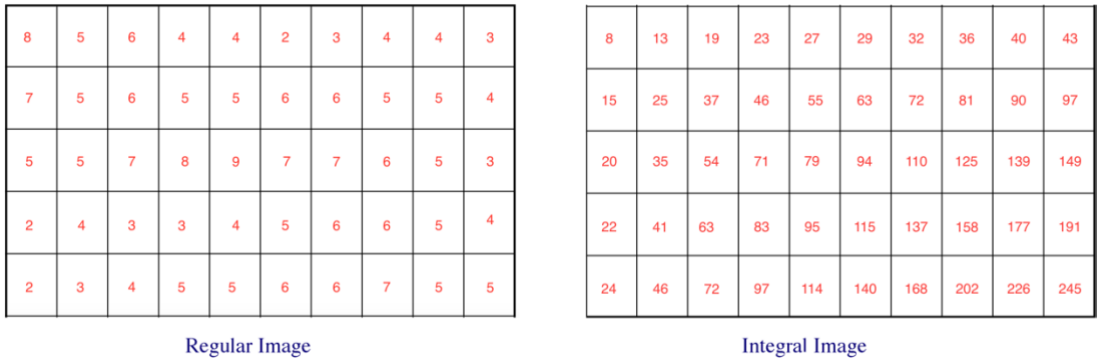

Figura 3. A l'esquerra, imatge original; a la dreta, imatge integral (Font: (7)).

Per tant, a partir de l'(Eq. 1 i la imatge original de la Figura 3, es pot extreure que el valor del punt $ii(2,2)$ es calcula de la següent manera:

$$ii(2,2) = i(1,1) + i(1,2) + i(2,1) + i(2,2) \quad \textbf{(Eq. 2)}$$

$$ii(2,2) = 8 + 7 + 5 + 5 = 25 \quad \textbf{(Eq. 3)}$$

Per determinar el valor d'una regió quadrada o rectangular només s'ha de consultar els valors de les seves cantonades de tal manera que:

$$v[(x_1', y_1'), (x_2', y_2')] = ii(x_2', y_2') - ii(x_1', y_2') + ii(x_1', y_1') - ii(x_2', y_1') \quad \textbf{(Eq. 4)}$$

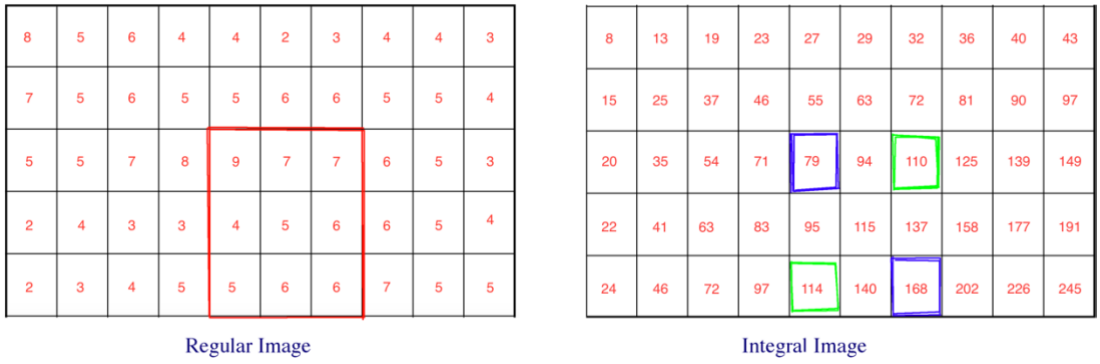

Figura 4. A l'esquerra, imatge original amb la zona a estudiar delimitada. A la dreta, la imatge integral amb els valors de l'equació ressaltats (Font: (7)).

A partir de l'(Eq. 4 i la Figura 4, el valor $v[(5,3),(7,5)]$ és:

$$v[(5,3),(7,5)] = ii(7,5) - ii(5,5) + ii(5,3) - ii(7,3) \quad \textbf{(Eq. 5)}$$

$$v[(5,3),(7,5)] = 168 - 114 + 79 - 110 = 23 \quad \textbf{(Eq. 6)}$$

3.2.1.3. Algoritme AdaBoost

Un dels factors claus en l'aprenentatge del classificador és l'algoritme AdaBoost. Aquest algoritme està dissenyat per determinar els falsos positius i negatius en les dades proporcionades. D'aquesta manera s'aconsegueix un classificador més precís (7).

Tenint una equació com l'(Eq. 7) cada f(x) representa una *feature* coneguda com a *weak classifier* o classificador feble. La part esquerra de la igualtat F(x) es coneix com un *strong classifier* o classificador fort, i correspon a la combinació de múltiples classificadors febles. Per tant, mentre que una *feature* pot no ser determinant una combinació d'elles sí.

$$F(x) = \alpha_1 f_1(x) + \alpha_2 f_2(x) + \alpha_3 f_3(x) + \dots$$ **(Eq. 7)**

Figura 5. Combinació de classificadors febles (Font: (7)).

A l'hora de provar una característica, es comprovarà quins encerts i quines errades ha comès, tot seguit, l'algoritme AdaBoost busca un reforç adaptatiu utilitzant una altra característica que complementi al classificador més fort obtingut fins ara. D'aquesta manera, augmenta la importància de les imatges en què s'ha equivocat l'anterior característica com a falsos negatius i per tant augmenta els pesos d'aquestes imatges en l'algoritme general. Un cop l'algoritme està optimitzat es passa a fer la detecció en cascada.

3.2.1.4. Classificador en cascada

Per tal d'augmentar la velocitat del classificador es duu a terme la classificació en cascada. Així doncs, en l'estudi d'una regió d'una imatge primer es mira si aquesta compleix la característica més forta, en el cas que es compleixi se segueix amb les següents característiques. En el moment que una de les característiques falla es considera que no s'ha detectat res i s'agafa una nova regió a estudiar. D'aquesta manera només les cares reals seran avaluades per l'algoritme complet (7).

3.2.2. *Deep learning*

El *deep learning* o aprenentatge profund forma part de la doctrina de *machine learning* dins del camp de la intel·ligència artificial. Una de les característiques principals del *deep learning* respecte a altres tipus d'intel·ligència artificial és l'aprenentatge a partir de representacions com poden ser imatges, vídeos o text, sense haver de regir-se per cap codi de normes introduït per un humà (8). El *deep learning*

intenta replicar l'aprenentatge i l'estructura neuronal d'un cervell humà. Aquesta estructura s'anomena xarxa neuronal (9).

3.2.2.1. Elements d'una xarxa neuronal

Les xarxes neuronals artificials tenen una inspiració en les xarxes neuronals biològiques, és a dir, els cervells. És per això que la seva estructura és similar i s'hi pot trobar molts paral·lelismes. Els components que formen una xarxa neuronal artificial són els següents.

3.2.2.1.1 Neurona

Les neurones o nodes tenen una evident similitud amb les neurones biològiques. Donats diferents estímuls o *inputs,* la neurona proporciona un únic valor de sortida o *output*. Aquest valor podrà ser l'estímul de neurones de les següents capes o el resultat final si se situa en l'última capa de la xarxa (10).

3.2.2.1.2 Pesos

Els pesos o *weights* determinen la importància de l'estímul rebut. Per exemple, una neurona que rep dos impulsos tindrà més en compte l'impuls que té un 0,8 de pes que el que té un pes de 0,2 (10).

3.2.2.1.3 Biaix

Cada neurona consta d'un biaix que se'l pot considerar un *offset*. Aquest biaix és un terme independent en la funció i garanteix que quan tots els estímuls rebuts siguin 0 hi hagi una activació predeterminada (10).

3.2.2.1.4 Funció d'activació

La funció d'activació és aquella que descriu el comportament de sortida d'una neurona. En funció de l'*input* rebut la funció d'activació determinarà si la neurona s'activa o no. Aquest resultat d'activació és el que es transmetrà a les següents neurones (11).

$$Y = \sum (weight * input) + bias \quad \textbf{(Eq. 8)}$$

3.2.2.1.5 Capes

En general, totes les xarxes neuronals estan estructurades en capes de neurones. La primera capa se l'anomena *input layer* i l'última se l'anomena *output layer*. Totes les capes de neurones que queden entre aquestes dues es diu que estan amagades o que són *hidden layers*.

Quantes capes, quantes neurones per cada capa, les connexions entre les neurones de cada capa, el tipus de neurones de cada capa... són alguns dels paràmetres diferencials que configuren cada tipus de xarxa neuronal.

3.2.2.2. Tipus de xarxes neuronals

El tipus de xarxa neuronal d'un model va relacionat amb l'estructura de capes: nombre de capes, nombre de neurones de cada capa, tipus de neurones de capes, connexions entre capes...

A causa de la gran quantitat de combinacions que hi pot haver d'aquests paràmetres existeixen moltes estructures adients per assolir diferents objectius.

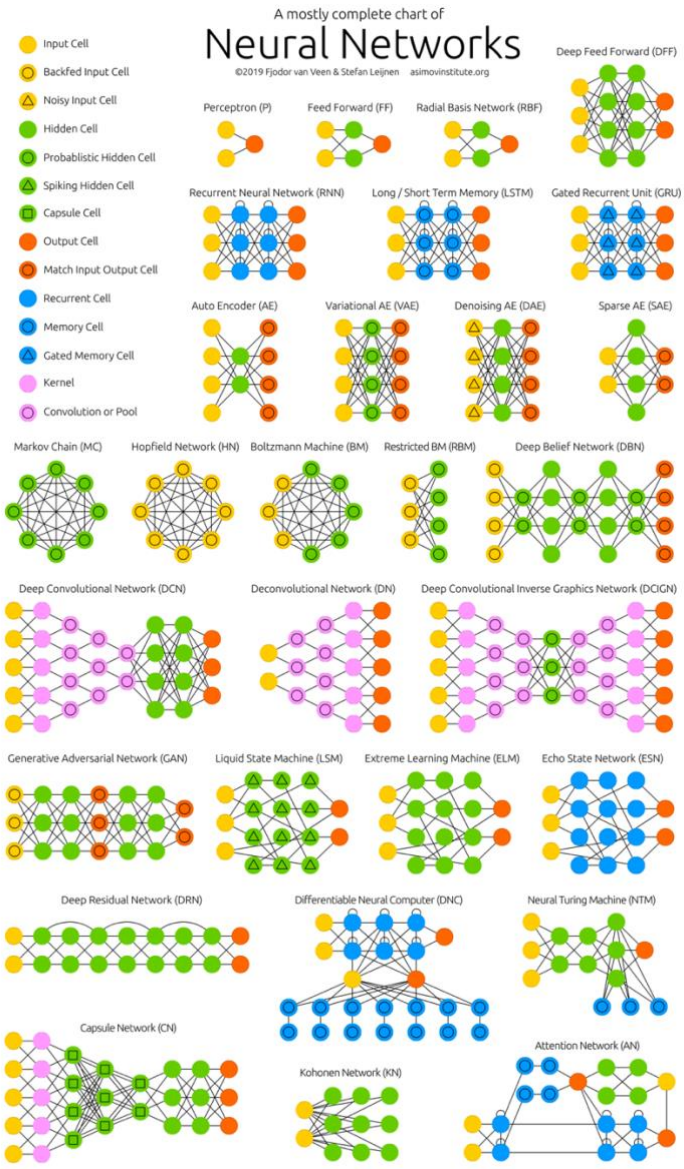

Figura 6. Estructures de xarxes neuronals (Font: (12)).

3.2.2.2.1 Convolucional

Per a l'aprenentatge amb imatges una de les estructures més populars és la xarxa neuronal convolucional (CNN). Aquest tipus de xarxa està inspirada en el còrtex visual d'un cervell biològic.

Aquest tipus de xarxes neuronals fan servir un kernel o filtre que emmagatzema informació de la imatge processada tot reduint dimensions. L'estructura del kernel es defineix a partir de criteris arbitraris per tal de recollir informació sobre els límits de les diverses regions de la imatge. Per a la Figura 7, veiem com el filtre es defineix de la següent manera:

1	0	1
0	1	0
1	0	1

Figura 7. Estructura del kernel

Per tant, si es multiplica aquesta màscara a cada regió 3x3 de la imatge s'obté el següent:

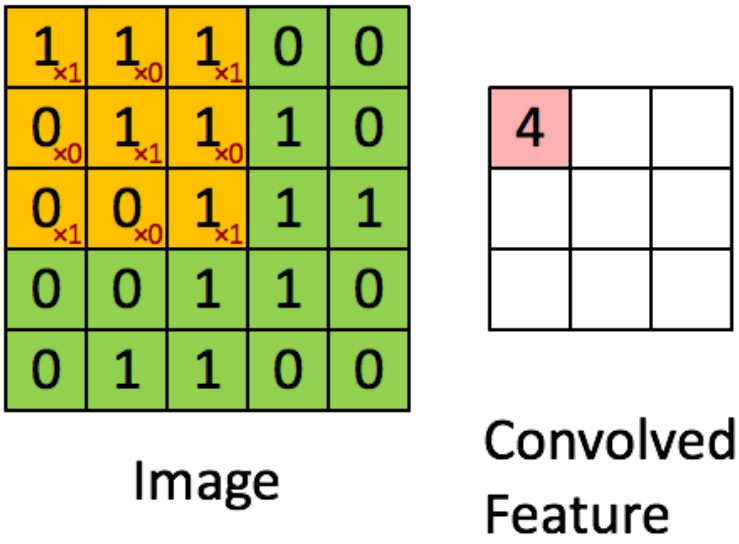

Figura 8. Característica resultant de filtrar la imatge original (Font: (13)).

Els filtres poden tenir 3 dimensions: alçada, amplada i profunditat. Un kernel 3x3x3 (Figura 9) seria una opció coherent per "convolucionar" una imatge RGB.

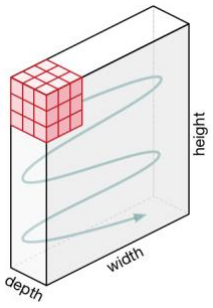

Figura 9. Kernel 3x3x3 (Font: (13)).

Després d'aquesta primera fase de filtratge, se segueix reduint la dimensió de les capes per optimitzar la *performance* del model. D'aquesta manera, diverses capes de *pooling* redimensionaran altra vegada les *features*. Aquest tipus de capes agrupen les regions a reduir de manera similar que les capes de filtratge, tanmateix, els criteris que es fan servir per aconseguir el valor final són diferents.

El valor de la regió pot anar condicionat per un criteri de màxims, en el que el valor és el valor màxim d'un dels components de la regió; o per un criteri de mitjanes, en el que el valor serà el valor mig de tots els components de la regió.

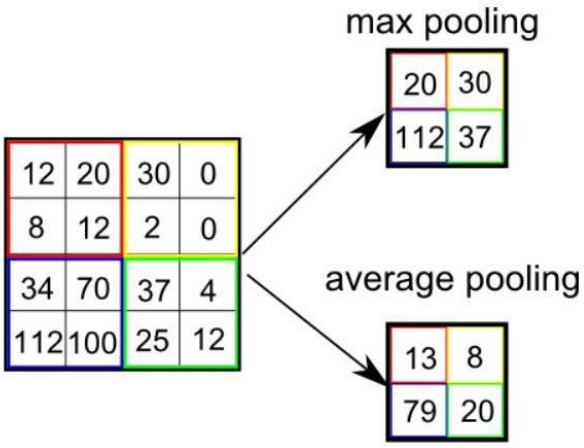

Figura 10. Max *pooling* vs average *pooling* (Font: (13)).

Les variants d'aquest tipus d'estructura més utilitzades per a la detecció d'objectes són la R-CNN, Fast-R-CNN i Faster-R-CNN. Tot i que cada una suposa una millora de l'anterior, presenten alguns problemes que fan que el seu ús no sigui indisputable.

Aquestes xarxes neuronals necessiten pel seu entrenament unes mides d'imatge entre els 600 i els 1.024 píxels, això fa que els *datasets* ocupin més que aquells destinats a entrenar-se amb mides de 200, 300 o 400 píxels. Aquest desavantatge pot ser significant en bases d'imatges amb moltes mostres i presentar problemes a l'hora de gestionar l'emmagatzematge (14).

Un dels altres problemes recurrents és la incapacitat de rendir a una velocitat de vídeo *feed*. Es donen casos en què aquestes xarxes neuronals no tenen la velocitat de predicció necessària per a mostrar resultats instantanis. Aquest defecte, en una disciplina en la qual les aplicacions en viu són molt importants i atractives, pot fer que aquest tipus de xarxes neuronals passin a un segon pla.

3.2.2.2.2 SSD (*Single Shot Detector*)

L'arquitectura *Single Shot Detector* fou creada per a obtenir millors *performances* en tasques de detecció d'objectes. Un estudi realitzat el mes de novembre del 2016 revelà que s'assoliren nous rècords de precisió enregistrant una *mean Average Precision* per sobre del 74% a 59 fps en *datasets* estàndards (15). *Single Shot* significa que la tasca de detecció d'objectes s'aconsegueix amb una sola passada per la xarxa neuronal.

L'arquitectura d'aquest tipus de xarxes neuronals està basada en l'arquitectura VGG-16. Tanmateix, les últimes *fully connected layers* s'intercanvien per unes capes convolucionals addicionals per tal de facilitar l'extracció de característiques a múltiples escales i reduir progressivament la mida de transmissió en les dades entre capes (16).

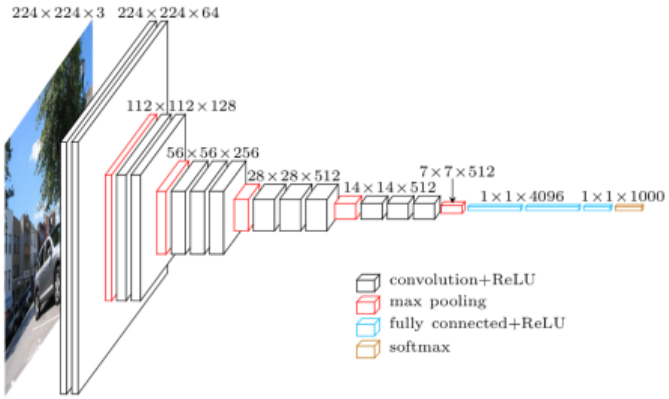

Figura 11. Estructura VGG-16 (Font: (16)).

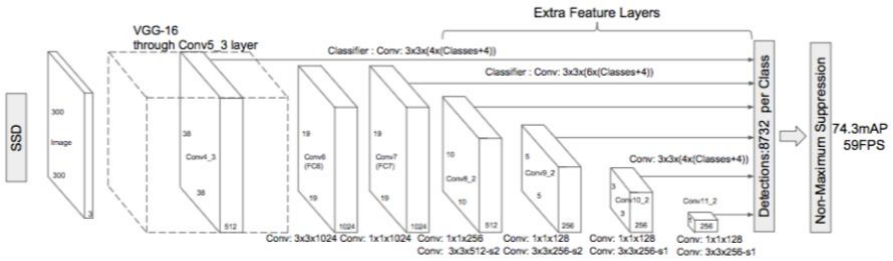

Figura 12. Estructura SSD (Font: (16)).

S'escollí l'estructura VGG-16, ja que aquesta proporciona un alt rendiment en tasques de classificació d'imatges d'alta qualitat i és popular pel seu bon rendiment en l'àmbit de *transfer learning*. El *transfer learning* és una pràctica molt freqüent en termes de detecció d'objectes per aprenentatge profund. Consisteix en el fet que un model entrenat per a una tasca específica serveixi com a punt de partida d'un entrenament independent, d'aquesta manera es reutilitza l'aprenentatge anterior i no es comença des de 0 (17). Existeix una àmplia varietat de models prèviament entrenats amb els quals començar a entrenar un model propi (18).

4. Software

4.1. C++

L'any 1983, el doctor Bjarne Stroustrup presenta una proposta millorada del llenguatge C que ell mateix havia enginyat 14 anys abans. En aquest llenguatge l'operador ++ comporta un increment de valor unitari en variables enteres, així doncs, aquest llenguatge en base C se l'anomenà C++.

C++ és un programa compilat, és a dir que compila directament en codi màquina i això el fa un dels programes més ràpids del món. En el moment de la seva creació, aquest llenguatge es considerà un llenguatge d'alt nivell, ja que la seva descripció estava força lluny del codi màquina final, tanmateix, amb l'aparició de llenguatges de molt més alt nivell que C++ ha quedat en terra de ningú entre l'alt i el baix nivell (19).

Afegit al seu gran rendiment, l'altre avantatge del llenguatge C++ és la seva portabilitat. La disponibilitat de múltiples compiladors del llenguatge permet executar programes en moltes plataformes diferents. Afegit a una àmplia varietat de llibreries portables això fa que C++ sigui un llenguatge molt polivalent.

La configuració d'un entorn estable per tal de compilar el programa cada vegada que es vulguin executar nous canvis és un dels inconvenients a tenir en compte. A més a més, C++ és un llenguatge "*strongly typed"*, terme que fa referència a aquells llenguatges que necessiten la definició del tipus de variables en ser aquestes declarades. Amb una gran quantitat de tipus de dades possibles, fa que el llenguatge no sigui intuïtiu en certs aspectes.

C++ està situat en el top 5 llenguatges més populars i utilitzats en l'actualitat (20).

4.2. Python

A finals de la dècada dels 80 Guido van Rossum conceptualitzà la idea del llenguatge de programació Python. L'informàtic dissenyà un llenguatge de *scripting*, interpretat, de senzilla sintaxi i evitant l'ús dels blocs de programació. Són aquests trets principals els que han dut a posicionar Python com el llenguatge de programació més popular de l'actualitat (21).

Python comporta una senzilla sintaxi basada en la sagnia del codi, facilitat per a la portabilitat i l'execució dels programes, a més a més d'una llarguíssima llista de llibreries molt potents que aporten compleció al llenguatge. Totes aquestes característiques el fan una eina molt bona per a la iniciació de la programació.

Un dels desavantatges evidents és el temps d'execució, ja que en ser un llenguatge interpretat aquest és significativament més gran que el temps d'execució en C++ per dos programes que realitzen la mateixa tasca. Aquest estudi centrat a generar totes les possibilitats de K-mers, unes seqüències de nucleòtids de 4 tipus diferents, demostra que per a la contemplació de tots els 13-mers (4^{13}) el programa amb Python és més de 25 vegades més lent que l'escrit amb C++ (22).

K-mer	Number of Combinations	Python Runtime (sec)	C++ Runtime (sec)	Ratio
13	67,108,864	61.23	2.42	25.30
14	268,435,456	224.07	8.38	26.74
15	1,073,741,824	919.24	31.62	29.07

Figura 13. Taula comparativa per a la generació de diferents K-mers, el temps d'execució en Python i en C++ (Font: (22)).

4.3. Tensorflow

Tensorflow és un software *open-source* impulsat per Google amb la finalitat de donar un suport ampli per a la creació i entrenament de models dins l'àmbit del *machine learning*. Adaptable a diferents nivells d'abstracció segons les habilitats i les necessitats de l'usuari, Tensorflow proporciona un alt nombre de possibilitats a desenvolupar. Per a l'interès del treball, es té en compte les diverses eines de detecció d'objectes així com la gran varietat de models prèviament entrenats a poder utilitzar (23).

4.4. OpenCV

OpenCV és una llibreria de visió per computador i *machine learning* escrita amb C++ que es pot utilitzar en C++, Python, Java i Matlab (24). És una eina essencial, ja que permet el tractament d'imatges amb bastant facilitat: proporciona opcions de segmentació, substitució, filtratge, representació... entre d'altres. A més a més, el fet que es pugui fer servir amb diversos llenguatges suposa una facilitat a l'hora d'escalar programes i per exemple, fer-los portables. Aquests trets fan que la llibreria OpenCV se situï per sobre d'altres paquets de processament d'imatges com Matlab o Octave, programes molt potents però amb poca portabilitat.

Una de les combinacions amb millor rendiment a hores d'ara per visió per computador és la combinació del llenguatge C++ amb la llibreria d'eines OpenCV.

4.5. Dlib

La llibreria Dlib és un conjunt d'eines C++ que conté múltiples algoritmes útils per al tractament de dades i imatges per a intel·ligència artificial (25). Una de les eines més interessants és la detecció d'una cara i la predicció de les seves faccions. La cara es divideix en 68 punts diferents, corresponent cada un a un punt determinat preestablert.

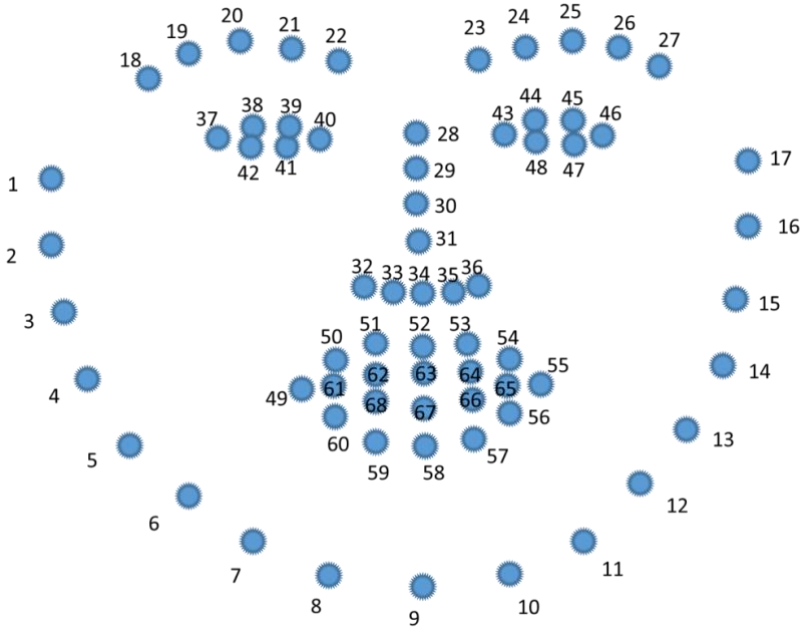

Figura 14. Esquema dels 68 punts facials predits per Dlib (Font: (26)).

Aquesta llibreria es pot utilitzar amb Python i C++.

4.6. LabelImg

LabelImg és una eina d'anotació d'imatges escrita amb Python amb primera publicació el setembre de l'any 2015. Aquesta classe d'eines permeten crear fitxers d'anotació que, referits a un fitxer visual, especifiquin els objectes que s'hi troben i on es localitzen en la imatge. LabelImg proporciona una anotació a fitxer XML en format ImageNet (27).

4.7. Codi propi

A partir dels softwares anteriorment comentats s'han desenvolupat diferents programes i eines per tal d'automatitzar processos o aconseguir l'objectiu directe del projecte. Moltes d'aquestes eines realitzen una tasca genèrica cosa que les fa útils en possibles futures situacions.

5. Implementació

5.1. Mètode de detecció

Per a dur a terme l'objectiu del treball és imprescindible escollir un mètode efectiu per aquest tipus de tasques. Moltes de les eines pioneres en el desenvolupament d'intel·ligència artificial consideren obsolets la detecció Viola-Jones així com altres algorismes pioners anys enrere. És per això que actualment aquestes eines deixen de suportar aquest tipus de pràctiques per tal d'enfocar el desenvolupament en l'aprenentatge profund. Davant l'oportunitat de realitzar la detecció en el marc de l'estat de l'art de la disciplina, en aquest projecte s'ha optat per treballar amb un model de xarxa neuronal SSD MobileNet V2 FPNLite 320x320 prèviament entrenat, adquirit del TensorFlow 2 Detection Model Zoo (18). Dins l'àmbit de la pràctica de detecció d'objectes amb TensorFlow, aquest tipus de xarxa neuronal SSD presenta alt rendiment per a deteccions ràpides.

L'objectiu de l'entrenament d'aquesta xarxa neuronal és la detecció de dues classes d'objectes: cara sense mascareta i cara amb mascareta. A més a més del seu encert analític es valora la seva usabilitat en una Webcam convencional a distàncies properes o mitjanes.

5.2. Confecció d'un *dataset*

Un dels elements essencials per establir bons resultats en la detecció d'objectes és l'obtenció d'una base d'imatges formada per un conjunt de prova i un conjunt d'entrenament. Dins d'aquests conjunts s'han de trobar els objectes a detectar pel model, en el cas que ocupa el projecte: dos objectes. Cada imatge ha d'estar anotada de tal manera que se sap quants objectes apareixen en la imatge, en quins píxels estan representats cada un i quina classe d'objecte són.

Les mostres proporcionades en un *dataset* han de garantir una diversitat dels objectes a detectar, ja que un mateix objecte pot tenir diverses formes, colors i tonalitats. També s'han de contemplar nivells d'il·luminació de l'escenari a tractar, mida dels objectes en la imatge, inclinació d'aquests... És per això que s'ha de formar una base d'imatges àmplia i diversa si es volen aconseguir grans resultats. D'altra banda, també és important que aquestes mostres no siguin extremadament complicades, ja que podrien dificultar la detecció final d'aquests objectes.

El *dataset* final s'ha confeccionat a partir de diferents subconjunts amb un total de 24.768 d'imatges.

A partir del *Face Mask Detection Dataset* obtingut del repositori Kaggle i prèviament anotat pels seus creadors, s'ha adquirit 4.326 imatges amb 5552 objectes de cares amb mascareta i 1.569 objectes de

cares sense mascareta. Aquesta base d'imatges, formada per imatges en situacions quotidianes i amb múltiples objectes per foto, aporta el vessant més difícil de la base d'imatges per a la detecció de mascaretes en cares humanes.

Figura 15. Exemples extrets del *Face Mask Detection Dataset* (Font: (28)).

D'altra banda, a partir del FFHQ *dataset* s'obtenen 70.000 mostres d'alta qualitat de rostres centrats i ben il·luminats extrets de la xarxa social Flickr, que serviran per entrenar la detecció de les cares sense mascareta. A més a més, es durà a terme un processament automatitzat sobre aquestes imatges que permetrà eixamplar la mostra de cares amb mascareta. Notar que no totes 70.000 imatges s'utilitzen en el conjunt final de la base d'imatges.

Figura 16. Exemples extrets del FFHQ *dataset* (Font: (29)).

Per últim, s'ha aportat una mostra simbòlica de 50 imatges realitzades en entorn personal i processades de la mateixa manera que les imatges del FFHQ *dataset*.

Figura 17. Exemples de la base dades personal.

A més a més d'aquestes imatges, s'ha preparat un *dataset* de mascaretes per poder generar noves cares amb mascareta. S'han segmentat múltiples mascaretes quirúrgiques, FFP2, FFP3 i de roba per 5 posicions diferents: decantat total a l'esquerra, decantat parcial a l'esquerra, centrat, decantat parcial a la dreta i decantat total a la dreta. S'ha augmentat el nombre d'aquestes mostres modificant colors i saturació de les mostres originals.

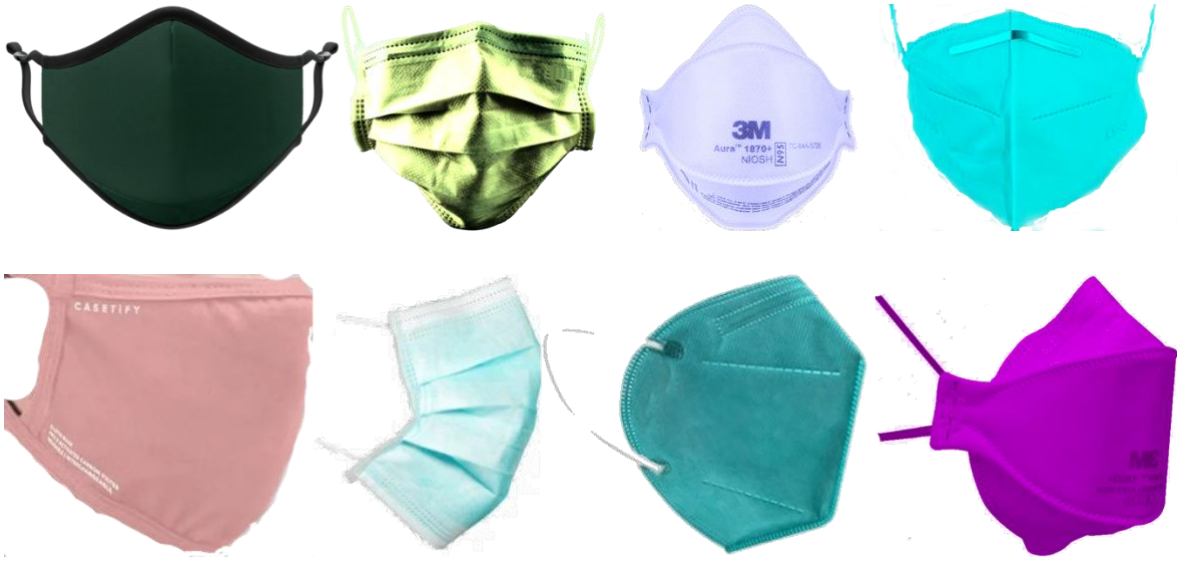

Figura 18. Exemples de mascaretes per a cares centrades i decantades a l'esquerra de la imatge.

5.2.1. Anotació d'objectes en imatges

L'anotació d'imatges es durà a terme en fitxers XML en format ImageNet. Aquesta mena de fitxer permet crear objectes d'informació a partir de *tags* que s'obren, <tag>, i es tanquen </tag>. Així doncs, múltiples etiquetes poden estar dins d'un *tag* "pare" que els conté. L'estructura d'etiquetes bàsica del format ImageNet és la següent:

```
<annotation>
    <folder></folder>
    <filename></filename>
    <path></path>
    <source>
        <database>Unknown</database>
    </source>
    <size>
        <width></width>
        <height></height>
        <depth></depth>
    </size>
```

```
13      <segmented>0</segmented>
14      <object>
15          <name></name>
16          <pose>Unspecified</pose>
17          <truncated>0</truncated>
18          <difficult>0</difficult>
19          <bndbox>
20              <xmin></xmin>
21              <ymin></ymin>
22              <xmax></xmax>
23              <ymax></ymax>
24          </bndbox>
25      </object>
26  </annotation>
```

Es pot veure que les tres primeres propietats dins del *tag annotation* refereixen a nom i ubicació de la imatge anotada. Més endavant, l'etiqueta *size* conté informació sobre les 3 dimensions de la imatge. Per últim, hi ha un exemple d'objecte buit, en què l'etiqueta *name* defineix la classe d'objecte que és i les propietats dins de l'etiqueta *bndbox* delimiten les coordenades de la cantonada superior esquerra i la cantonada inferior dreta de l'objecte dins la imatge. En el cas de conviure més d'un objecte en una imatge, la mateixa estructura d'*object* pot ser repetida en el mateix nivell tantes vegades com objectes hi hagi.

S'emmagatzemen aquestes 26 línies sota el nom del fitxer dist.xml, que servirà com a model d'anotació per a les eines d'anotació automàtica que es desenvoluparan en els propers apartats.

A més a més, es generaran dos tipus d'anotacions: una que emmarcarà tota la cara i una que emmarcarà només boca i nas, en definitiva, la zona de la mascareta. D'aquesta manera es brinda més possibilitats d'experimentació per a un millor rendiment del model final.

5.2.1.1. Anotació manual

Per aquelles imatges que no es poden etiquetar de manera automàtica, s'ha de dur a terme un etiquetatge manual fent ús del programa LabelImg. No es poden automatitzar les anotacions de les imatges pròpies fetes amb mascareta, ja que justament no existeix una eina que detecti cares amb mascareta.

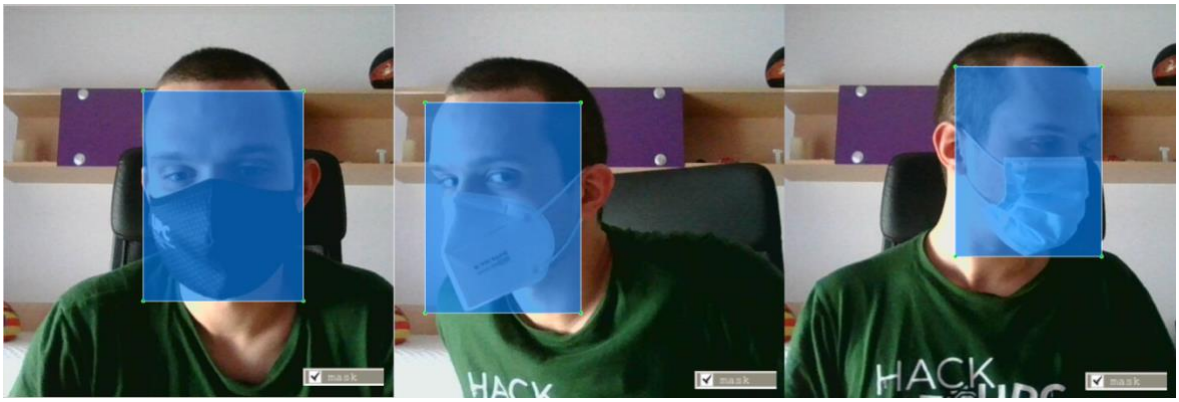

Figura 19. Exemples d'etiquetatge des del programa LabelImg.

5.2.1.2. JSON a XML

El *Face Mask Detection Dataset* aporta anotacions en fitxers JSON. Aquest format no satisfà l'estructura del projecte, però sí que aporta molt bona informació sobre els objectes de les imatges. Amb l'objectiu de traduir aquesta informació a format XML ImageNet, s'ha ideat una eina json_to_xml.py que sostraurà les propietats necessàries del format JSON i les escriurà en XML.

D'aquesta manera, una anotació en JSON vista d'aquesta manera:

```
{
  "FileName": "1801.jpg",
  "NumOfAnno": 1,
  "Annotations": [
    {
      "isProtected": false,
      "ID": 924868908868875136,
      "BoundingBox": [451, 186, 895, 697],
      "classname": "face_no_mask",
      "Confidence": 1,
      "Attributes": {}
    }
  ]
}
```

El programa l'escriurà en XML d'aquesta altra:

```
<annotation>
    <folder>json_images</folder>
    <filename>1801.jpg</filename>
```

```
4       <path>/json_images/1801.jpg</path>
5       <source>
6          <database>Unknown</database>
7       </source>
8       <size>
9          <width>1386</width>
10         <height>1385</height>
11         <depth>3</depth>
12      </size>
13      <segmented>0</segmented>
14      <object>
15         <name>no-mask</name>
16         <pose>Unspecified</pose>
17         <truncated>0</truncated>
18         <difficult>0</difficult>
19         <bndbox>
20            <xmin>451</xmin>
21            <ymin>186</ymin>
22            <xmax>895</xmax>
23            <ymax>697</ymax>
24         </bndbox>
25      </object>
26   </annotation>
```

5.2.2. Generar cares amb mascareta

Per a generar automàticament cares amb mascaretes a partir d'un conjunt de cares, primer s'han de detectar les cares i localitzar les faccions més importants per al propòsit: orelles, nas i barbeta. Gràcies a la llibreria Dlib i al seu *face landmark detection* s'aconsegueix detectar una cara i descriure-la en 68 punts estratègics diferents. En la Figura 20 s'aprecia una de les imatges de la base de dades FFHQ processada per la llibreria en qüestió i mostrant els 68 punts esmentats. Són d'interès els punts 2 i 16 per localitzar les orelles, el 9 per localitzar la barbeta i el 30 per assegurar que la mascareta es col·locarà per sobre del nas.

Figura 20. A la dreta, imatge original; a l'esquerra, imatge processada per Dlib.

El fitxer generate_masks.py és un programa original escrit amb Python que a partir de rostres sense mascareta genera noves imatges de cada un dels rostres amb una mascareta de roba, una quirúrgica, una FFP2 i una FFP3. Una estructura de carpetes vàlida hauria de ser similar a la següent:

```
generate_masks/
├─ facial_landmarks/
│  └─ shape_predictor_68_face_landmarks.dat
├─ xml/
│  └─ dist.xml
├─ generate_masks.py
└─ functions.py

masks/
├─ c/
│  ├─ clothee/
│  │  └─ ...
│  ├─ ffp2/
│  │  └─ ...
│  ├─ ffp3/
│  │  └─ ...
│  └─ surgical/
│     └─ ...
├─ i/
```

```
│   ├── clothee/
│   │   └── ...
│   ├── ffp2/
│   │   └── ...
│   ├── ffp3/
│   │   └── ...
│   └── surgical/
│       └── ...
├── lc/
│   ├── clothee/
│   │   └── ...
│   ├── ffp2/
│   │   └── ...
│   ├── ffp3/
│   │   └── ...
│   └── surgical/
│       └── ...
├── r/
│   ├── clothee/
│   │   └── ...
│   ├── ffp2/
│   │   └── ...
│   ├── ffp3/
│   │   └── ...
│   └── surgical/
│       └── ...
└── rc/
    ├── clothee/
    │   └── ...
    ├── ffp2/
    │   └── ...
    ├── ffp3/
    │   └── ...
    └── surgical/
        └── ...

faces/
```

```
└─ ...

output/
├─ ann/
│  ├─ face/
│  ├─ mask/
│  └─ masked_face/
└─ img/
```

A les primeres línies de codi s'importen aquells paquets i llibreries que s'utilitzen en el cos del codi. Notar que en la línia 3 s'afegeixen totes aquelles funcions pròpies que es criden al llarg del codi, declarades al fitxer functions.py situat al mateix nivell que generate_masks.py.

```
1  # import the necessary packages
2  from __future__ import print_function
3  from functions import rotate_image, get_masks, get_face, xml_masked_face, xml_mask, xml_face
4  import numpy as np
5  import argparse
6  import imutils
7  import dlib
8  import cv2
9  import os
10 import os.path
11 from os import listdir
12 import random
13 import sys
14 import time
15 from etaprogress.progress import ProgressBar
```

Els arguments d'entrada necessaris per córrer el programa són els següents:

- shape-predictor: ruta al shape_predictor_68_face_landmarks.dat.
- input_dir: ruta a la carpeta contenidora de les imatges amb rostres.
- mask_dir: ruta a la carpeta contenidora de les mascaretes.
- output_dir: ruta a la carpeta on s'escriuran les imatges generades.
- output_ann: ruta a la carpeta on s'escriuran les anotacions de les imatges generades.

A més a més, hi ha dos arguments d'entrada opcionals que són els següents:

- show: *True* si es vol veure els resultats imatge a imatge a mesura que es van generant. *False per defecte.*
- ckpt: Si es vol començar a partir d'un fitxer determinat. Per defecte s'agafen tots.

```
17  # construct the argument parser and parse the arguments
18  ap = argparse.ArgumentParser()
19  ap.add_argument("-p", "--shape-predictor", required=True,
20      help="path to facial landmark predictor")
21  ap.add_argument("-i", "--input_dir", required=True,
22      help="path to input dir")
23  ap.add_argument("-m", "--mask_dir", required=True,
24      help="path to mask dir")
25  ap.add_argument("-o", "--output_dir", required=True,
26      help="path to output dir")
27  ap.add_argument("-oa", "--output_ann", required=True,
28      help="path to output annotations dir")
29  ap.add_argument("-s", "--show", required=False,
30      help="show results (False by defalut)")
31  ap.add_argument("-c", "--ckpt", required=False,
32      help="checkpoint")
33  args = vars(ap.parse_args())
```

Per aquests arguments i per a l'estructura anteriorment definida es pot córrer el programa executant en el terminal:

```
~/generate_masks$ python generate_masks.py \
      -p facial_landmarks/shape_predictor_68_face_landmarks.dat \
      -i ../faces/ -m ../masks/ -o ../output/img/ -oa ../output/ann/
```

En les línies 37 i 38 es prepara el detector de cares i predictor dels punts facials a partir de la llibreria Dlib. A la línia 41 es crida a la funció *get_masks(mask_path)* del fitxer functions.py, que retorna un diccionari ordenat de Python amb totes les matrius de les imatges corresponents, ordenades per posició i tipus de mascareta. D'aquesta manera, es carreguen les imatges tan sols una vegada i s'optimitza el procés. Entre les línies 43 i 47 es llisten els noms de les imatges a processar i s'escurça si el programa s'ha executat amb l'argument "ckpt" a partir del fitxer en posició del nombre entrat. Per últim, en la línia 49 s'inicialitza una barra de progrés que informarà a través del terminal el percentatge de tasca realitzada i el temps estimat per a la seva finalització.

```
35  # initialize dlib's face detector (HOG-based) and then create
36  # the facial landmark predictor
```

```
37  detector = dlib.get_frontal_face_detector()
38  predictor = dlib.shape_predictor(args["shape_predictor"])
39
40  # get masks
41  masks = get_masks("/home/ignasi/Documents/TFG/datasets/alpha_masks/prod/")
42
43  files = listdir(args["input_dir"])
44  files.sort()
45
46  if args["ckpt"]:
47      files = files[int(args["ckpt"]):]
48
49  bar = ProgressBar(len(files), max_width=100)
```

A partir de la línia 51 es recorren tots elements de la llista *files*, és a dir, es recorrerà cada imatge a processar i es repetirà el *loop* tantes vegades com imatges a tractar. De la línia 52 a la 54 s'actualitza l'índex que indica el progrés a la barra d'estat en el terminal. En la 57 i 58, es carrega la imatge i es transforma a escala de grisos amb l'ajuda de la llibreria OpenCV (cv2). Es detecten les cares de la imatge a partir del detector prèviament inicialitzat i la imatge en escala de grisos a la línia 61. L'operació retorna una llista de rectangles que enquadren cada una de les cares detectades, si no s'ha detectat cap cara, la imatge ja no té interès i a la línia 64 salta tot el procés per llegir la següent imatge.

```
51  for i, file in enumerate(files):
52      bar.numerator = i + 1
53      print(bar, end='\r')
54      sys.stdout.flush()
55
56      # load the input image and convert it to grayscale
57      img1 = cv2.imread(args["input_dir"] + file)
58      gray = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
59
60      # detect faces in the grayscale image
61      rects = detector(gray, 1)
62
63      if len(rects) < 1:
64          continue
```

Tot seguit, es crida a la funció *get_face(rects, predictor, gray)* que retornarà un diccionari amb molta informació essencial de la cara més gran detectada en la imatge. Les claus d'aquest diccionari són les següents:

- *position*: pot prendre els valors "c", "l", "lc", "r" o "rc" en funció de la decantació de la cara en la imatge. Aquest paràmetre és necessari per poder escollir les mascaretes que més s'adapten a aquesta orientació.
- *x_offset*: paràmetre que correspon a la coordenada x més a l'esquerra en què se situarà la mascareta, localitzant el segon punt de la predicció Dlib coincidint amb l'orella que queda a l'esquerra de la imatge.
- *y_offset*: paràmetre que correspon a la coordenada y més amunt en què se situarà la mascareta, localitzant el punt número 30 de la predicció Dlib per garantir que la mascareta se situarà per sobre el nas.
- *width*: amplada calculada des de *x_offset* fins a la x de l'orella dreta (punt 16).
- *height*: alçada calculada des de *y_offset* fins a la y de la barbeta (punt 9).
- *angle*: angle d'inclinació, en graus, entre la y del punt 16 i la y del punt 2. D'aquesta manera es determina la inclinació entre orelles i s'inclina la mascareta en funció d'aquest paràmetre.
- *topleft*: punt de dalt a l'esquerra de la cara sencera.
- *botright*: punt de baix a la dreta de la cara sencera.

Tota aquesta informació s'emmagatzema a la variable *face,* que serà un paràmetre necessari a l'hora de cridar la funció *xml_face(imgs_path, ann_path, file, face, shape)* que generarà l'anotació per a les fotos amb cara descoberta a partir dels punts *topleft* i *botright*. Aquest tipus d'objectes se'ls anota amb el nom "no-mask".

```
66    face = get_face(rects, predictor, gray)
67    xml_face(args["input_dir"], args["output_ann"], file.split(".")[0], face, img1.shape)
```

A partir de la línia 69 s'inicia un altre bucle que recorrerà tots els tipus de mascareta especificats per aquella posició. Per tal de conservar la imatge original, es duplica aquesta per ser processada en la variable *img_out*. Per cada tipus de mascareta, s'escull una de les opcions a l'atzar, es redimensiona per l'amplada i l'alçada calculades prèviament i s'inclina per l'angle especificat entre les línies 72 i 76. Notar que no es té en compte mantenir cap ràtio de la imatge de la mascareta original.

```
69    for type in masks[face["position"]].keys():
70        img_out = img1
71        #take one random mask for each type
72        img2 = random.choice(masks[face["position"]][type])
73
74        # resize and rotate
```

```
75        img2 = cv2.resize(img2, (face["width"], face["height"]))
76        img2 = rotate_image(img2, face["angle"])
77
78        # to numpy arrays
79        img_out = np.array(img_out)
80        img2 = np.array(img2)
```

Un cop obtinguda tota aquesta informació, es recorrerà tota la regió d'interès on s'ha de col·locar la mascareta per sobre i se substituirà cada píxel pel nou que correspongui. En aquest doble bucle que recorrerà simultàniament la imatge de la mascareta i la imatge de la cara, les variables (x1, y1) representaran el píxel de la imatge de la cara mentre que el píxel de la imatge de la mascareta estarà representat per (x2, y2). En la línia 87 es mira el component alpha del píxel actual de la mascareta, en cas que aquest sigui 0 o proper a 0, voldrà dir que és transparent i que s'haurà de mantenir l'antic píxel de la imatge original. La substitució o no substitució es duu a terme amb l'equació de la línia 88.

```
82        # all pixels that are going to be replaced
83        for (y2, y1) in enumerate(range(0 + face["y_offset"], face["height"] + face["y_offset"])):
84            for (x2, x1) in enumerate(range(0 + face["x_offset"], face["width"] + face["x_offset"])):
85
86                # if alpha = 0 old pixel remains
87                alpha = round(img2[y2][x2][3]/255)
88                img_out[y1][x1] = img_out[y1][x1] * (1-alpha) + img2[y2][x2][:3]*alpha
```

Tenint ja la imatge processada emmagatzemada en la variable *img_out*, resta escriure les anotacions dels objectes de les imatges amb les funcions *xml_masked_face(imgs_path, ann_path, file, face, shape)* i *xml_mask(imgs_path, ann_path, file, face, shape, type)*. La primera d'aquestes funcions anotarà les cares senceres mentre que la segona anotarà només la zona de la cara que té mascareta. A més a més, en la línia 95 s'escriurà la nova imatge processada amb mascareta, i, si s'ha executat el programa amb l'argument "show", es mostrarà cada nova imatge gràcies a l'acció de la línia 98. La línia 99 suposa el final dels bucles inicialitzats a les línies 51 i 69, mentre que la línia 100 netejarà l'estat del terminal un cop finalitzats tots els processos.

```
90        out_filename = file.split(".")[0] + "_" + face["position"] + "_" + type
91        xml_masked_face(args["output_dir"], args["output_ann"], out_filename, face, img_out.shape)
92        xml_mask(args["output_dir"], args["output_ann"], out_filename, face, img_out.shape, type)
93
94        # write the output image with the mask
95        cv2.imwrite(args["output_dir"] + out_filename + ".png", img_out)
96
97        if args["show"]:
```

```
98              cv2.imshow("Output", img_out)
99              cv2.waitKey(0)
100 print()
```

Així doncs, per la imatge de la Figura 20 s'obtenen les següents imatges:

Figura 21. Imatges generades de cares amb mascareta.

Altres exemples:

Figura 22. Imatges generades de cares amb mascareta.

Figura 23. Imatges generades de cares amb mascareta.

5.2.3. Col·locació aleatòria de cares sobre fons aleatoris

Per tal de proporcionar diversitat de posicions i de mides de les cares en una imatge, s'idea una eina que donats un conjunt d'imatges etiquetades i un conjunt d'imatges de fons es redimensiona i es col·loca aleatòriament cada cara en un fons aleatori. El conjunt d'imatges de fons s'extreu del repositori de Kaggle Stanford *Background Dataset* (30). Una estructura vàlida per executar el programa és la següent:

```
labeled_on_background/
└─ labeled_on_background.py
```

```
input/
├─ ann/
│  └─ ...
└─ img/
   └─ ...

output/
├─ ann/
│  └─ ...
└─ img/
   └─ ...
background/
└─ ...
```

Aquesta eina, emmarcada en el fitxer labeled_to_backgorund.py utilitza les següents dependències:

```python
# import the necessary packages
from __future__ import print_function
from etaprogress.progress import ProgressBar
import sys
import time
import os
import cv2
import numpy as np
import os.path
from os import listdir
import argparse
import xml.etree.ElementTree as ET
import random
```

Els arguments necessaris per executar el procés són els següents:

- input_dir: ruta a la carpeta contenidora de les imatges etiquetades.
- output_dir: ruta a la carpeta on s'escriuran les imatges etiquetades sobre fons aleatoris.
- input_xml: ruta a la carpeta contenidora de les anotacions originals.
- output_xml: ruta a la carpeta on s'escriuran les noves anotacions.
- background_dir: ruta a la carpeta contenidora dels fons.

D'altra banda, també n'hi ha d'opcionals:

- from: a partir de quina posició dins la carpeta d'imatges es vol aplicar el procés.
- to: fins quina posició dins la carpeta d'imatges es vol aplicar el procés.

```
15  # construct the argument parser and parse the arguments
16  ap = argparse.ArgumentParser()
17  ap.add_argument("-i", "--input_dir", required=True,
18     help="path to input dir")
19  ap.add_argument("-o", "--output_dir", required=True,
20     help="path to output dir")
21  ap.add_argument("-ix", "--input_xml", required=True,
22     help="path to input xml dir")
23  ap.add_argument("-ox", "--output_xml", required=True,
24     help="path to outpùt xml dir")
25  ap.add_argument("-b", "--background_dir", required=True,
26     help="path to background dir")
27  ap.add_argument("-f", "--from", required=False,
28     help="from file")
29  ap.add_argument("-t", "--to", required=False,
30     help="to file")
31  args = vars(ap.parse_args())
32
33  PATH_TO_INPUT = args["input_dir"]
34  PATH_TO_OUTPUT = args["output_dir"]
35  PATH_TO_INPUT_XML = args["input_xml"]
36  PATH_TO_OUTPUT_XML = args["output_xml"]
37  PATH_TO_BACKGROUND = args["background_dir"]
```

Per aquests arguments i per a l'estructura de carpetes anteriorment suggerida, es pot executar el procés des del terminal de la següent manera:

```
~/labeled_on_background $ python labeled_on_backround.py \
      -i ../input/img/ -o ../output/img/ -ix ../input/ann/ \
      -ox ../output/ann/ -b ../background/
```

De les línies 39 a la 50, es llisten totes les imatges a processar, i s'escurça la llista en funció dels arguments opcionals "to" i "from". També es llisten totes les imatges que són s'utilitzaran de fons. Per últim, s'inicialitza la barra de procés de les tasques.

```
39  files = listdir(PATH_TO_INPUT)
```

```
40  files.sort()
41
42  bgs = listdir(PATH_TO_BACKGROUND)
43  bgs.sort()
44
45  if args["to"]:
46      files = files[:int(args["to"])]
47  if args["from"]:
48      files = files[int(args["from"]):]
49
50  bar = ProgressBar(len(files), max_width=100)
```

Per cada imatge amb cares a processar, es carrega la imatge en qüestió, la seva anotació en XML i una imatge de fons aleatòria.

```
52  for i, file in enumerate(files):
53      bg = cv2.imread(os.path.join(PATH_TO_BACKGROUND, random.choice(bgs)))
54      face = cv2.imread(os.path.join(PATH_TO_INPUT, file))
55      tree = ET.parse(os.path.join(PATH_TO_INPUT_XML, file.split(".")[0] + ".xml"))
56      root = tree.getroot()
57
58      bg_shape = bg.shape
```

S'itera per cada un dels objectes anotats per cada cara, sigui quina sigui la seva categoria, i es llegeix les seves coordenades. La primera posició de la llista *hw* representa l'alçada de l'objecte mentre que la segona posició representa l'amplada. A partir de les dimensions de la imatge de fons, a la línia 67 es genera una nova alçada per l'objecte que representarà entre un 20% i un 80% de l'alçada total, pel que fa a la nova amplada, serà entre un 60% i un 100% de l'amplada que seria proporcional a la nova alçada. D'aquesta manera es deformaran breument algunes cares i s'aconseguiran resultats més genèrics a l'hora d'entrenar. També es genera un nou punt aleatori (*nx*, *ny*) que correspondrà a la posició de la cara en la imatge de fons.

```
59      for bb in root.iter('bndbox'):
60          xmin = int(bb.find('xmin').text)
61          ymin = int(bb.find('ymin').text)
62          xmax = int(bb.find('xmax').text)
63          ymax = int(bb.find('ymax').text)
64          hw = [ymax - ymin, xmax - xmin]  ##original face dimensions
65
66          # new face height (20-80% of background height)
```

```
67          nh = random.randint(int(0.2*bg_shape[0]), int(0.8*bg_shape[0]))
68          # new face width (60-100% proportioned respect height)
69          nw = int(random.uniform(0.6,1)*hw[1]*nh/hw[0])
70
71          ny = random.randint(0, bg_shape[0] - nh) # random y
72          nx = random.randint(0, bg_shape[1] - nw) # random x
```

Un cop obtinguda i calculada tota la informació necessària, es procedeix a adquirir la matriu que correspon a la regió d'interès de l'objecte, en aquest cas, la cara a sobreposar (línia 75). En la línia 78 es redimensiona la imatge a les noves dimensions calculades i a la 80 se substitueixen els píxels de la imatge de fons pels emmagatzemats a la *roi_face*. Per últim, s'actualitzen els valors de les noves coordenades en l'objecte d'anotació.

```
74          # get face from whole original image
75          roi_face = face[ymin:ymax, xmin:xmax, :]
76
77          # resize face to calcd dimensions
78          roi_face = cv2.resize(roi_face, (nw, nh))
79          # subsitute new face in calcd position
80          bg[ny:(ny+nh), nx:(nx+nw), :] = roi_face
81
82          bb.find('xmin').text = str(nx)
83          bb.find('ymin').text = str(ny)
84          bb.find('xmax').text = str(nx + nw)
85          bb.find('ymax').text = str(ny + nh)
```

Havent sortit del *loop* iniciat a la línia 59, ja es pot escriure la nova imatge en el directori de sortida així com el nou fitxer d'anotacions (línies 87 i 88).

```
87      cv2.imwrite(os.path.join(PATH_TO_OUTPUT + file), bg)
88      tree.write(os.path.join(PATH_TO_OUTPUT_XML, file.split(".")[0] + ".xml"))
89
90      bar.numerator = i + 1
91      print(bar, end='\r')
92  print()
```

Figura 24. Exemples d'imatges de cares sobreposades en un fons aleatori.

5.2.4. Modificació aleatòria de la brillantor de la imatge

Per completar la *data augmentation* del *dataset* és imprescindible variar la brillantor de les imatges per poder entrenar amb diverses il·luminacions. Això s'aconseguirà a partir del programa brigthness_da.py, que a partir d'un conjunt d'imatges i amb l'ajuda del fitxer functions.py, retornarà cada una de les imatges retocades amb una brillantor aleatòriament diferent. En aquest cas, les anotacions no es modifiquen de cap manera i els fitxers que serveixen per a les imatges d'entrada també serviran per a les imatges de sortida. L'estructura de carpetes pot ser similar a la següent:

```
brightness_da/
├─ brightness_da.py
└─ functions.py

input/
└─ ...

output/
```

Les llibreries i els paquets utilitzats en el programa són els següents. Notar que en la línia 10 s'importa la funció *get_random_bright(img)* que es declara en el fitxer functions.py, situat al mateix nivell que el programa principal:

```
1  # import the necessary packages
2  from __future__ import print_function
3  from etaprogress.progress import ProgressBar
4  import sys
5  import os
6  import cv2
7  import os.path
```

```
8   from os import listdir
9   import argparse
10  from functions import get_random_bright
```

Els arguments d'entrada necessaris per executar el procés són els següents:

- input_dir: ruta a la carpeta contenidora de les imatges a processar.
- output_dir: ruta a la carpeta on s'escriuran les imatges resultants.

D'altra banda, també n'hi ha d'opcionals:

- from: a partir de quina posició dins la carpeta d'imatges es vol aplicar el procés.
- to: fins quina posició dins la carpeta d'imatges es vol aplicar el procés.

```
12  # construct the argument parser and parse the arguments
13  ap = argparse.ArgumentParser()
14  ap.add_argument("-i", "--input_dir", required=True,
15      help="path to input dir")
16  ap.add_argument("-o", "--output_dir", required=True,
17      help="path to output dir")
18  ap.add_argument("-f", "--from", required=False,
19      help="from file")
20  ap.add_argument("-t", "--to", required=False,
21      help="to file")
22  args = vars(ap.parse_args())
23
24  PATH_TO_INPUT = args["input_dir"]
25  PATH_TO_OUTPUT = args["output_dir"]
```

A partir dels arguments d'entrada definits i de l'estructura de carpetes anteriorment proposada, es pot executar el programa per terminal de la següent manera:

```
~/brightness_da $ python brightness_da.py -i ../input/ -o ../output/
```

Entre les línies 27 i 35, es llisten les imatges a modificar i s'inicialitza la barra de progrés.

```
27  files = listdir(PATH_TO_INPUT)
28  files.sort()
29
30  if args["to"]:
```

```
31      files = files[:int(args["to"])]
32  if args["from"]:
33      files = files[int(args["from"]):]
34
35  bar = ProgressBar(len(files), max_width=100)
```

Entrem en un bucle repetit tantes vegades com imatges a processar hi ha. En les línies 38 i 39 s'actualitza l'estat de la barra de progrés, mentre que a la línia 40 es carrega la imatge a processar. A la línia 42 se sobreescriuen els valors emmagatzemats en la variable *img* pels retornats de la funció *get_random_bright(img)*, que proporciona ja la imatge processada amb brillantor aleatòria. Finalment, la línia 43 escriu la nova imatge a la carpeta de sortida.

```
37  for i, file in enumerate(files):
38      bar.numerator = i + 1
39      print(bar, end='\r')
40      img = cv2.imread(os.path.join(PATH_TO_INPUT, file))
41
42      img = get_random_bright(img)
43      cv2.imwrite(os.path.join(PATH_TO_OUTPUT + file), img)
44
45  print()
```

Però, com s'aconsegueix l'aleatorietat de brillantor? La vertadera "màgia" passa a partir de la crida de *get_random_bright(img)* i al fitxer functions.py.

La Figura 26 representa l'arbre de decisions que suposa aquesta tasca. Hi ha una primera decisió aleatòria que decidirà si es retoca tota la imatge conjunta, a files, a columnes, a quadrats o a diagonals. La primera possibilitat generarà una matriu d'1 amb les dimensions de la imatge a retocar. Es multiplicaran els 1 per un enter aleatori entre el 50 i el 150, suposant això la intensitat del retoc final. Notar que un 0 suposaria la no modificació i el 255 la màxima brillantor o foscor, depenent de si s'afegeix o es resta aquesta matriu a la imatge original. Aquesta decisió també es pren aleatòriament, si s'afegeix resultarà una imatge amb més brillantor i si es resta resultarà amb més foscor.

La resta de possibilitats s'ajuden de la funció *get_bands(img)*, aquesta funció generarà una matriu a partir de *n_cols* submatrius, sent *n_ cols* un enter aleatori entre 1 i 9. Cada submatriu representa una fila de la imatge, aquestes podran prendre aleatòriament el valor 0 (no modificació) o un aleatori entre el 50 i el 150, resultant així distribucions desiguals de retoc de la imatge.

Les columnes, quadrats i diagonals també s'ajuden de la funció *rotate_img(image, angle)*. Per al primer d'aquests tres casos, es gira 90 graus la matriu de files que prové de *get_bands* i s'obté una matriu de

columnes. Per als quadrats, se suma la matriu files i la matriu columnes convertint així la matriu en una espècie de tauler d'escacs. Per últim, la matriu girada 45 graus suposarà la inclinació diagonal de les franges.

Figura 25. Exemples resultants del procés brightness_da.py

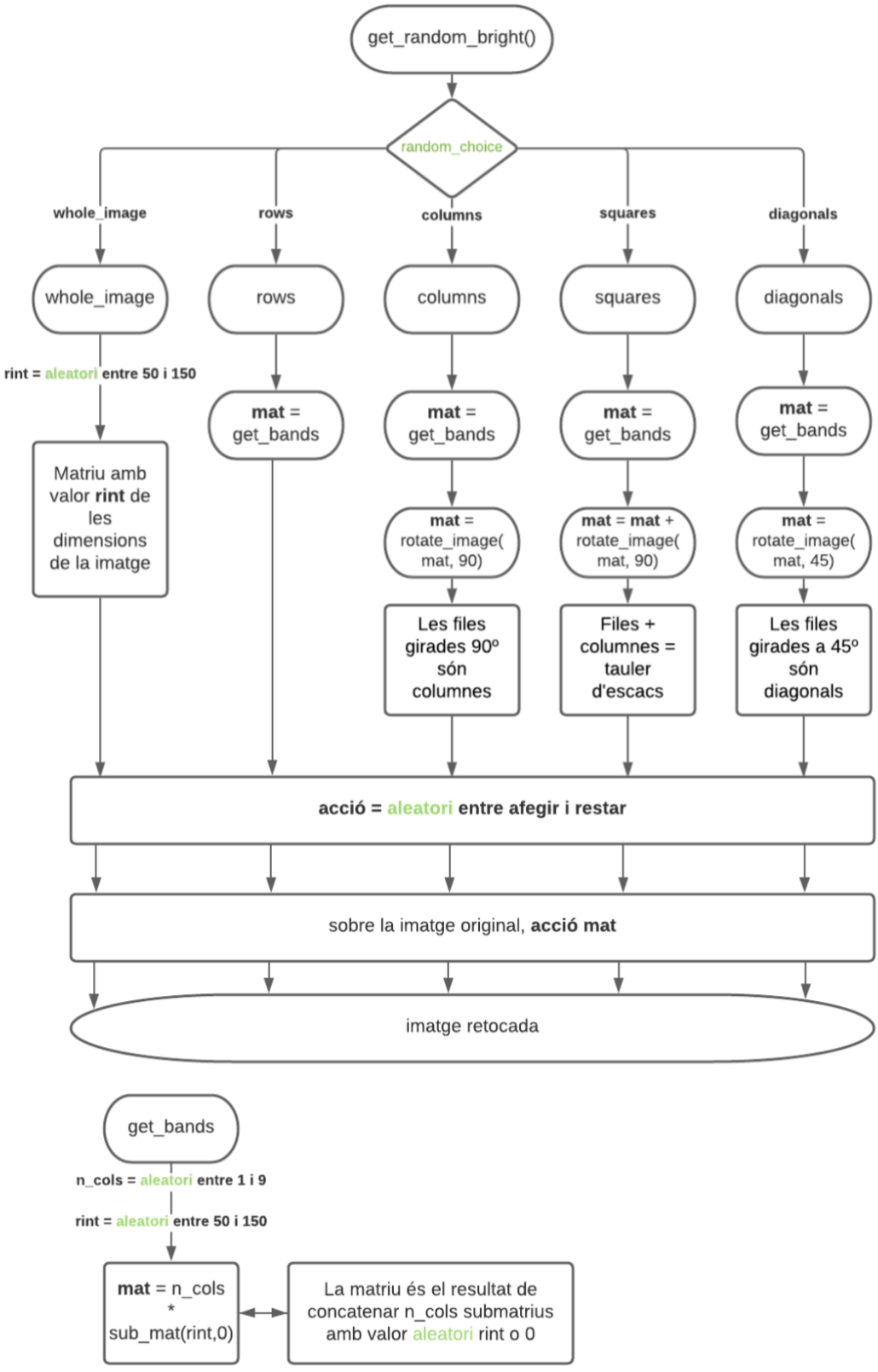

Figura 26. Esquema de decisions i processos per la funció *get_random_bright(img)*

5.2.5. Conjut de prova i d'entrenament

Un cop complet tot el *dataset*, cal crear a un conjunt de prova i un conjunt d'entrenament a partir d'aquestes imatges anotades. La ràtio estàndard és de 0,9, és a dir, que el conjunt d'entrenament representa un 90% de les mostres mentre que el de prova el 10% restant (31). Per a la partició de la base d'imatges és interessant que la distribució sigui aleatòria mantenint la ràtio especificada, d'aquesta manera s'assegura un entrenament més genèric. Per a realitzar aquesta tasca s'ha ideat el programa test_train.py, que a partir de la següent estructura de carpetes, arguments d'entrada i execució de comandes dividirà el conjunt d'entrada en un 90% de *train* i un 10% de *test*, ajuntant així en les mateixes carpetes els arxius d'anotació i els d'imatge, necessari per a l'apartat 5.2.6.

```
train_test/
└─ train_test.py

input/
├─ ann/
└─ img/

output/
├─ test/
└─ train/
```

Arguments d'entrada requerits:

- input_dir, i: ruta a la carpeta contenidora de les imatges etiquetades.
- output_dir, o: ruta a la carpeta contenidora de les carpetes test/ i train/ per a les imatges.
- input_xml, ix: ruta a la carpeta contenidora de les anotacions.
- output_xml, ox: ruta a la carpeta contenidora de les carpetes test/ i train/ per a les anotacions.

Arguments d'entrada opcionals:

- ratio, r: valor de la ràtio sobre 1. Per defecte 0,9.
- from, f: a partir de quina posició dins la carpeta d'imatges es vol aplicar el procés.
- to, t: fins a quina posició dins la carpeta d'imatges es vol aplicar el procés.

Execució:

```
~/train_test $ python train_test.py -i ../input/img/ -o ../output/ \
      -ix ../input/ann/ -ox ../output/
```

5.2.6. Imatges i anotacions a TFRecords

Per tal de dur a terme l'entrenament de la xarxa neuronal, TensorFlow proposa l'ús del format TFRecord a l'hora d'importar els conjunts de prova i d'entrenament. El format TFRecord fa servir *Protocol Buffers* per serialitzar o desserialitzar la informació i emmagatzemar-la en bytes. D'aquesta manera, en un mateix fitxer s'agrupen totes les mostres de cada conjunt a més a més d'ocupar menys espai (32). Per convertir imatges i anotacions a TFRecord es pot utilitzar l'eina generate_tfrecord.py aportada per la documentació de TensorFlow Object Detection.

5.2.7. Unió de TFrecords

Una bona praxi en la creació dels TFRecords és tenir un fitxer per cada un dels *datasets* i conjunts de *train* i *test*. D'aquesta manera es poden combinar *datasets* en una base d'imatges més àmplia i trobar la millor combinació sense perdre les anotacions i imatges prèviament generades.

Per mantenir aquesta bona praxi però acabar tenint un TFRecord conjunt s'ha creat l'eina merge_tfrecord.py, que donada una carpeta contenidora dels fitxers TFRecord a unificar escriu el fitxer unit en la ruta d'*output* especificada.

5.3. Configuració prèvia a l'entrenament

Per a l'entrenament d'un model prèviament entrenat amb TensorFlow Object Detection s'ha d'adaptar la configuració original del model a les necessitats actuals d'entrenament. Per això, del fitxer pipeline.config original s'ha de tenir en compte la possible modificació dels següents paràmetres:

- num_classes: nombre de classes a detectar, en aquest cas, 2.
- batch_size: nombre de mostres que es propagaran a la vegada per la xarxa neuronal. Com més gran sigui aquest nombre més memòria lliure haurà de tenir l'ordinador en què s'executa l'entrenament.
- fine_tune_checkpoint: ruta al checkpoint del model prèviament entrenat.
- fine_tune_checkpoint_type: com que es vol entrenar una detecció, s'ha d'especificar "*detection*".
- label_map_path: ruta al fitxer en el qual s'especifiquen les etiquetes de les classes a detectar.
- input_path(train_input_reader): ruta al TFrecord d'entrenament.
- input_path(eval_input_reader): ruta al TFrecord de prova.

El label_map.pbtxt, és un fitxer que especifica les etiquetes de les classes a detectar, d'aquesta manera, aquest fitxer per al model entrenat en el projecte és el següent:

```
item {
    id: 1
    name: 'mask'
}

item {
    id: 2
    name: 'no-mask'
}
```

5.4. Entrenament

Per dur a terme l'entrenament d'un model s'utilitza l'eina de TensorFlow model_main_tf2.py, que necessitarà especificacions sobre la carpeta de sortida del model resultant, la seva configuració i el nombre de *steps* d'entrenament. Aquest últim paràmetre especifica quantes iteracions d'entrenament realitzarà el procés. Com més iteracions faci més ocasions hi haurà per augmentar la precisió del model. També s'ha de tenir en compte que un model amb excessives iteracions pot arribar a sobre entrenar-se i només detectar el conjunt d'imatges aportat i no imatges de casos reals, aquest fenomen s'anomena *overfit*.

L'eina TensorBoard proporciona informació sobre l'evolució de l'entrenament que s'està duent a terme. Aquest tauler mostra diferents paràmetres que permeten valorar la millora de l'entrenament. El paràmetre principal que TensorFlow proporciona per comprovar l'evolució de l'entrenament és la pèrdua o *total loss*. Aquest paràmetre és el valor que la xarxa neuronal té com a objectiu minimitzar, per tant, com més petit sigui el seu valor més precisió tindrà la xarxa neuronal. Segons indicacions de TensorFlow, el valor de pèrdua s'ha de situar per sota d'1. Sovint, un indicador de bon entrenament és quan aquest paràmetre disminueix exponencialment acabant per estancar-se en un valor força reduït.

A més a més de la *total loss* és interessant la informació proporcionada per la *classificaiton loss* i la *localization loss*. El primer d'aquests dos valors refereix a l'error a l'hora de classificar els objectes trobats, el segon, aporta informació sobre la precisió a l'hora d'emmarcar els objectes, és a dir, a l'hora de delimitar la *bounding box*.

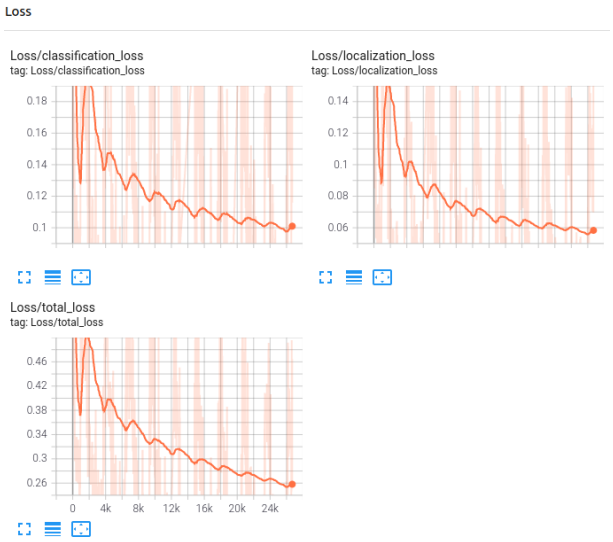

Figura 27. Corbes suavitzades de *classification loss, localization loss* i *total loss* a la plataforma Tensorboard.

6. Possibles aplicacions

L'objectiu del treball era desenvolupar un sistema de detecció de mascaretes en cares que permeti el seu ús en un entorn controlat a temps real, les aplicacions per aquesta eina poden anar des d'ajuda a persones amb visibilitat reduïda a ser part d'un sistema automatitzat controlador d'entrada en un edifici, sala, despatx, establiment...

Mentre no s'aconsegueixi la immunitat de grup de la COVID 19 serà obligatori l'ús de la mascareta per a totes les persones, fins i tot les que ja estan vacunades. Una de les aplicacions que es consideren és que sigui una eina per a una persona amb visibilitat reduïda que treballa de cara al públic. D'aquesta manera, amb el seu ordinador de feina i una càmera pot assegurar-se de que la persona que està atenent duu correctament posada la mascareta. Per a aquesta aplicació, s'hauria d'implementar un sistema d'avís acústic.

Una de les altres aplicacions seria un sistema de porteria automàtic, ja sigui per entrar a espais que es vol que siguin especialment segures o edificis oficials. Les portes automàtiques que tan comunes són avui en dia en qualsevol establiment, haurien de validar que a part del retorn de l'infraroig que emeten també s'està detectant una cara amb mascareta. Aquesta aplicació podria córrer tant en un ordinador de sobretaula de personal de seguretat com en un sistema incrustat a partir d'una placa computacional.

Per últim, es podria instal·lar aquest sistema juntament amb una pantalla que mostrés el resultat de la detecció en un pas públic com un carrer concorregut o un centre comercial. La visió per computador és una disciplina molt atractiva per la gent perquè el resultat del seu procés és visual i molt fàcil d'entendre per a tothom, d'aquesta manera la gent podria "jugar" a fer funcionar la detecció i acabaria marxant amb la mascareta posada.

7. Projecte com a punt de partida

A part de les aplicacions que estarien força a prop de poder ser comercialitzades comentades en l'apartat anterior, el projecte proporciona altres factors de valor com són totes les eines creades per a tractaments de *datasets*.

En el camp del *deep learning* sovint es necessita treballar amb grans quantitats de fitxers i els GigaBytes que suposen. Les interfícies d'usuari que habitualment s'utilitzen poden estancar-se a l'hora de llistar i gestionar aquests fitxers, és per això que les eines desenvolupades en aquest projecte poden ser de gran ús per a futurs projectes dependents d'una bona creació de base d'imatges.

S'han desenvolupat eines àmpliament genèriques que es poden utilitzar per a qualsevol ús com moure o copiar fitxers en massa, llistar i comptar totes les diferents extensions de fitxers que hi ha en una carpeta, anomenar fitxers en massa...

D'altra banda, s'han creat eines molt genèriques pel que fa a la producció de bases de dades com col·locació d'objectes ja etiquetats en imatges aleatòries, modificació aleatòria de la brillantor, redimensionar imatges en massa, escapçar imatges en massa, passar anotació XML a anotació TXT per a detector Viola Jones, concatenar imatges horitzontalment o canviar noms dels objectes anotats en massa.

A més a més, s'han desenvolupat eines genèriques per a l'ús i l'avaluació de TensorFlow com generar matriu de confusió, provar detecció de models en càmera web i en imatges o unió de TFRecords.

Per últim, també s'han generat eines més específiques però essencials per al projecte com és la generació de cares amb mascareta a partir de cares sense mascareta. És una eina més complexa, amb bona estructura de codi i possiblement adaptable a altres necessitats en un futur.

Més enllà de les eines creades per tal d'assolir el propòsit del treball també queda una herència o un exemple a seguir a l'hora de realitzar projectes per a detecció d'objectes.

8. Anàlisis de l'impacte ambiental

Els dispositius en els quals es pot executar el programa preparat són ordinadors, portàtils o plaques computacionals; elements que comparteixen una estructura electrònica molt similar i que l'impacte de la seva manufactura i el seu ús són similars. A més a més, també es requereix una càmera web per captar les imatges.

Pel que fa als portàtils i als dispositius electrònics en general, la majoria de les emissions CO2 de la vida d'aquests aparells es produeix en la seva manufactura. La raó d'aquesta significant dominància d'emissió en aquest àmbit és la producció asiàtica de gran part d'aquests dispositius. Un país tan important com la Xina genera un 67% de la seva energia a partir d'energies no renovables, sobretot carbó. Estimant un temps mitjà d'ús entre 3 i 4 anys, aquest període suposa un quart de les emissions totals en la vida de l'aparell, com més verd sigui l'origen de l'energia que es consumeix per al seu ús més petita serà aquesta petjada.

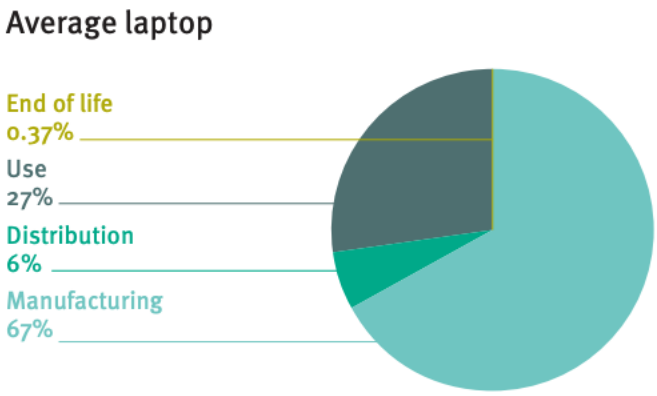

Figura 28. Emissions CO2 en la vida d'un portàtil (Font: (33)).

D'altra banda, la precocitat generalitzada en canviar de dispositius suposa que els residus electrònics tinguin un impacte negatiu si no es gestionen de manera correcta. Un 93% dels materials d'un ordinador són reciclables, però sovint la manera dels usuaris de desfer-se d'aquests dispositius no ajuden al fet que es realitzi un reciclatge dels seus materials, només un 6,5% dels dispositius es reciclen mitjançant els canals oficials que garanteixen un reciclatge local i maximitzat (34). Un ordinador està compost per metalls, materials ceràmics o vidres, plàstics i els components de la bateria. Són materials que en general es reciclen quotidianament i que fins i tot, diversos materials que formen la bateria poden reciclar-se. L'any 2014, es generaren 41,8 tones de residus electrònics, quantitat que augmenta un 5% cada any. Aquest mateix any, el valor d'aquests residus s'estimà en 48 bilions d'euros. Això conclou que amb un sistema de reciclatge electrònic més efectiu no només es podria minimitzar la

petjada dels nous dispositius a manufacturar, si no que també s'hi podria fer un comerç rendible de la gestió d'aquests residus.

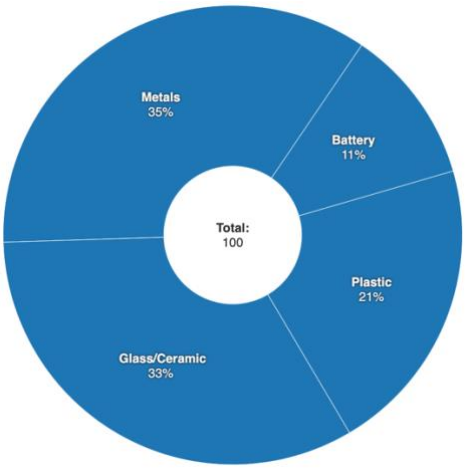

Figura 29. Composició d'un ordinador (Font: (34)).

L'altra variable de què dependrà l'impacte ambiental del *hardware* utilitzat per a l'execució del programa és la seva longevitat. La mitjana d'ús d'aquests dispositius són 3,11 anys, amb un any més de vida s'estalvia un 19% de carboni i un 18% d'aigua. Per a 4 anys més de vida s'estalvia entre un 30 i 45% de carboni i un 41% d'aigua, depenent de si el dispositiu s'han dut a terme algunes reparacions com per exemple el reemplaçament de la pantalla i de la bateria. Aquestes dades conclouen que des del punt de vista de l'impacte ambiental és rendible substituir certs elements deteriorats dels dispositius abans de comprar-ne de nous.

	Years of use	Embodied carbon per cent saved	Primary energy per cent saved	Water consumption per cent saved
Laptops				
Baseline	3.11	0%	0%	0%
1 year life extension	4.11	19%	n/a	18%
Longer economic lifetime	7.00	44%	n/a	41%
Longer life with screen & battery replacement	7.00	31%	n/a	n/a

Figura 30. Comparativa de recursos estalviats respecte al temps d'ús d'un portàtil (Font: (33)).

En termes generals, la confecció d'un ordinador suposa l'ús de 240 kg de combustible, 22 kg de substàncies químiques i 1500 litres d'aigua (34).

Pel que fa a les càmeres web, els seus components principals són polímers, vidre i materials ceràmics, policarbonats, coure i altres metalls. Amb una constitució similar amb els altres dispositius electrònics

comentats anteriorment, comparteixen els afers de reciclatge i de minimització de la seva petjada ecològica. La següent taula mostra, entre 8 marques diferents de càmeres, quines proposen més mètodes verds a l'abast dels consumidors.

	Axis	Bosch	Hikvision	Lilin	Mobotix	Panasonic	Sony	Vivotek
Energy efficient products	●					●		●
Reducing CO2 emissions	●	●			●	●		
Promoting a greener brand	●	●				●		●
Packaging	●							●
Reducing hazardous materials	●			●		●	●	●
Recycling	●					●		●

Figura 31. Taula comparativa entre marques i les seves accions i ajudes verdes pel consumidor (Font: (35)).

Conclusions

En aquest Treball de Fi de Grau s'ha desenvolupat una eina que detecta cares amb i sense mascareta en una entrada de vídeo en directe.

El desenvolupament en TensorFlow, Opencv, Python i C++, proporciona adaptabilitat multiplataforma, ús en diferents tipus de dispositius i funcionament sense necessitat de connexió a la xarxa. S'ha entrenat una xarxa neuronal a partir del model prèviament entrenat SSD MobileNet V2 FPNLite 320x320.

S'ha aconseguit una detecció genèrica per a tota mena de trets facials diversos gràcies a la creació d'una base d'imatges profunda i diversa. A partir de cares sense mascareta s'ha pogut automatitzar un procés de col·locació de mascaretes sobre aquestes cares, mascaretes de diferents tipus, colors i contrasts.

Sobre aquesta base d'imatges, s'han dut a terme uns processos automatitzats de *data augmentation* que a més a més d'augmentar el nombre de mostres han proporcionat robustesa a l'eina a l'hora de detectar en diferents tipus d'il·luminacions i mides i col·locació de les cares en la imatge.

Malgrat aquests avenços, una detecció exitosa dependrà d'un entorn controlat i ben il·luminat. Per a una mostra d'imatges més acotada per aconseguir el funcionament d'una xarxa neuronal en un entorn específic, s'aconseguiria un millor rendiment. Tanmateix, si el que es valora és un ús genèric en diferents entorns i estalviant processos nous d'entrenament aquesta eina és una opció vàlida.

Anàlisi Econòmica

Es calcularà el preu de desenvolupament de l'eina per tal de cobrir les despeses generades posant per cas que aquesta eina ha estat desenvolupada per una persona assalariada que treballa a una consultoria.

El temps emprat en realitzar la tasca es pot quantificar a partir del nombre de crèdits ECTS que constitueixen el Treball de Fi de Grau: 24. Cada crèdit suposa 25 hores de feina, per tant, el temps total empleat són 600 hores totals. De mitjana, un enginyer júnior a Espanya cobra 26.200 €/any de salari brut treballant a jornada completa de 8 hores al dia (36), aquest preu s'ha de multiplicar per 1,34, representant un 34% de seguretat social. L'any 2021 compta amb 251 dies laborables, 22 d'aquests són vacances pagades per al treballador. Per tant, els diners que cobra per hora treballada serien:

$$\frac{26.200\ € \cdot 1{,}34}{251 - 22\ dies} = 153{,}31 \frac{€}{dia} \quad \textbf{(Eq. 9)}$$

$$153{,}31 \frac{€}{dia} \cdot \frac{1\ dia}{8\ hores} = 19{,}16 \frac{€}{hora} \quad \textbf{(Eq. 10)}$$

A partir d'aquest preu/hora es pot calcular la despesa que es destinarà a pagar el salari brut del treballador:

$$19{,}16 \frac{€}{hora} \cdot 600\ hores = 11.496\ € \quad \textbf{(Eq. 11)}$$

A més a més, cal contemplar despeses de *hardware* i recursos utilitzats, com poden ser l'amortització del dispositiu utilitzat així com la llum, la xarxa...

Per a un ordinador de 1000 € sense IVA, amb 16 GB de RAM, 1 TB de memòria SSD i un processador amb targeta gràfica, es poden considerar 4 anys d'amortització. També s'afegeixen 300 € d'oficina, pantalla, teclat, ratolí, escriptori i cadira pels mateixos anys d'amortització. Això implica que el preu que s'ha d'amortitzar cada mes és el següent:

$$1300 \frac{€}{4\ anys} \cdot \frac{1\ any}{12\ mesos} = 27{,}09 \frac{€}{mes} \quad \textbf{(Eq. 12)}$$

Considerant que 600 hores de feina suposen aproximadament 4 mesos, el preu a pagar per amortització de *hardware* i material d'oficina és:

$$27{,}09 \frac{€}{mes} \cdot 4\ mesos = 108{,}34\ € \quad \textbf{(Eq. 13)}$$

Per últim, s'afegeixen 100 € d'ús de recursos periòdics com són la llum o la xarxa per connectar-se a internet durant 4 mesos.

Tipus de despesa	**Preu (€)**
Salari	11.496
Material	108,34
Recursos	100
Total	**11.704,34 €**

Taula 1. Suma de les despeses generades.

11.704,34 € és el preu total que cobreix les despeses generades a l'hora de realitzar aquesta tasca.

Des del punt de vista empresari, s'hauria de modelitzar un model de negoci que permet recuperar aquests diners i proporcionar un marge econòmic a l'empresa. Una opció seria arribar a molts compradors i així poder rebaixar el preu per client, o trobar un client que sigui una empresa molt gran que necessiti aquesta eina en molts llocs de treball.

També apareix la possibilitat d'oferir un manteniment i així cobrar ingressos passius per una quantitat d'hores més reduïda.

Bibliografia

1. World Health Organization. *Coronavirus Prevention* [en línia]. WHO, 2021. [Consulta: 7 juny 2021]. Disponible a: <https://www.who.int/health-topics/coronavirus#tab=tab_2>
2. Corporació Catalana de Mitjans Audiovisuals, S.A. *Notícies | Primer concert massiu en pandèmia: 5.000 persones amb tests d'antígens i mascareta FFP2* [en línia]. Actualitzat 29 març 2021. [Consulta: 7 juny 2021]. Disponible a: <https://www.ccma.cat/324/prova-de-foc-per-als-concerts-massius-el-sant-jordi-amb-5-000-persones-en-pandemia/noticia/3086049/>
3. Ilija Mihajlovic. *Everything You Ever Wanted To Know About Computser Vision* [en línia]. Publicat 25 abril 2019. [Consulta: 8 juny 2021]. Disponible a: <https://towardsdatascience.com/everything-you-ever-wanted-to-know-about-computer-vision-heres-a-look-why-it-s-so-awesome-e8a58dfb641e>
4. Paul Viola i Michael Jeffrey Jones. *Rapid Object Detection using a Boosted Cascade of Simple Features* [en línia]. Publicat febrer 2001. [Consulta: 8 juny 2021]. Disponible a: <https://www.researchgate.net/publication/3940582_Rapid_Object_Detection_using_a_Boosted_Cascade_of_Simple_Features>
5. Wikimedia Commons. *File:VJ featureTypes.svg* [en línia]. Publicat 4 octubre 2010. [Consulta: 8 juny 2021]. Disponible a: <https://commons.wikimedia.org/wiki/File:VJ_featureTypes.svg>
6. Analytics Vidhaya. *What are Haar Features used in Face Detection?* [en línia]. Publicat 12 novembre 2019. [Consulta: 8 juny 2021]. Disponible a: <https://medium.com/analytics-vidhya/what-is-haar-features-used-in-face-detection-a7e531c8332b>
7. Rohan Gupta. *Breaking Down Facial Recognition: The Viola-Jones Algorithm* [en línia]. Publicat 7 agost 2019. [Consulta: 8 juny 2021]. Disponible a: <https://towardsdatascience.com/the-intuition-behind-facial-detection-the-viola-jones-algorithm-29d9106b6999>
8. arm. *Glossary | Deep Learning vs Machine Learning* [en línia]. [Consulta: 8 juny 2021]. Disponible a: <https://www.arm.com/glossary/deep-learning-vs-machine-learning?utm_term=&gclid=CjwKCAjw47eFBhA9EiwAy8kzNOzYiKyVoP_36UzWOfr0aah8zOo6iDVcXOJm6Q5BKDBZS_ZC3-OYWxoC8FwQAvD_BwE>
9. Raúl Arrabales. *Deep Learning, qué es y por qué va a ser una tecnología clave en el futuro de la inteligencia artificial* [en línia]. Actualitzat 28 octubre 2016. [Consulta: 8 juny 2021]. Disponible a: <https://www.xataka.com/robotica-e-ia/deep-learning-que-es-y-por-que-va-a-ser-una-tecnologia-clave-en-el-futuro-de-la-inteligencia-artificial>
10. Bill Wilson. *The Machine Learning Dictionary* [en línia]. Actualitzat 25 juny 2012. [Consulta: 8 juny 2021]. Disponible a: <http://www.cse.unsw.edu.au/~billw/mldict.html>
11. Daniel Kobran i David Banys. *AI Wiki | Weights and Biases* [en línia]. Actualitzat 2020. [Consulta: 8 juny 2021]. Disponible a: <https://docs.paperspace.com/machine-learning/wiki/weights-and-biases>

12. The Asimov Institute. *Neural network zoo* [en línia]. Publicat 14 setembre 2016. [Consulta: 26 maig 2021]. Disponible a: <https://www.asimovinstitute.org/neural-network-zoo/>
13. Sumit Saha. *A Comprehensive Guide to Convolutional Neural Netoworks* [en línia]. Publicat 15 desembre 2018. [Consulta: 26 maig 2021]. Disponible a: <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>
14. Derek Chow. *Stack Overflow | Image size for object detection API* [en línia]. Publicat 26 setembre 2017. [Consulta: 5 maig 2021]. Disponible a: <https://stackoverflow.com/questions/46414472/image-size-for-object-detection-api/46415915#46415915>
15. Eddie Forson. *Understanding SSD Multibox – Real-time Object Detection In Deep Learning* [en línia]. Publicat 18 novembre 2017. [Consulta: 28 maig 2021]. Disponible a: <https://towardsdatascience.com/understanding-ssd-multibox-real-time-object-detection-in-deep-learning-495ef744fab>
16. Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg. *SSD: Single Shot Multibox Detector* [en línia]. Revisat 29 desembre 2016. [Consulta: 28 maig 2021]. Disponible a: <https://arxiv.org/abs/1512.02325>
17. Jason Brownlee. *A Gentle Introduction to Transfer Learning for Deep Learning* [en línia]. Publicat 20 desembre 2017. [Consulta 28 maig 2021]. Disponible a: <https://machinelearningmastery.com/transfer-learning-for-deep-learning/>
18. TensorFlow, Vighnesh Birodkar. *Tensorflow 2 Object Detection Model Zoo* [en línia]. Actualitzat 7 maig 2021. [Consulta: 8 juny 2021]. Disponible a: <https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md>
19. Usuari *Albatross* cpluscplus.com. *A Brief Description* [en línia]. [Consulta: 30 maig 2021]. Disponible a: <https://www.cplusplus.com/info/description/>
20. TIOBE Software BV. *TIOBE Index for June 2021* [en línia]. Actualitzat juny 2021. [Consulta: 30 maig 2021]. Disponible a: <https://www.tiobe.com/tiobe-index/>
21. Sohom Pramanick. *History of Python* [en línia]. Actualitzat 6 maig 2019. [Consulta: 30 maig 2021]. Disponible a: <https://www.geeksforgeeks.org/history-of-python/>
22. Naser Tamimi. *How Fast Is C++ Compared to Python?* [en línia]. Publicat 16 desembre 2020. [Consulta: 30 maig 2021]. Disponible a: <https://towardsdatascience.com/how-fast-is-c-compared-to-python-978f18f474c7>
23. TensorFlow. *Why TensorFlow* [en línia]. [Consulta: 1 juny 2021]. Disponible a: <https://www.tensorflow.org/about>
24. OpenCV team. *About* [en línia]. [Consulta: 1 juny 2021]. Disponible a: <https://opencv.org/about/>
25. Dlib. *Dlib C++ Library* [en línia]. Modificat 28 març 2021. [Consulta: 1 juny 2021]. Disponible a: <http://dlib.net/>

26. Sefik Ilkin Serengil. *Facial Landmarks for Face Recognition with Dlib* [en línia]. Publicat 20 nomvembre 2020. [Consulta: 5 maig 2021]. Disponible a: <https://sefiks.com/2020/11/20/facial-landmarks-for-face-recognition-with-dlib/>
27. Usuari *tztutalin* a github.com. *LabelImg* [en línia]. Actualitzat 1 maig 2021. [Consulta: 20 abril 2021]. Disponible a: <https://github.com/tzutalin/labelImg>
28. Wobot Intelligence. *Face Mask Detection Dataset* [en línia]. Actualitzat 2020. [Consulta: 6 abril 2021]. Disponible a: <https://www.kaggle.com/wobotintelligence/face-mask-detection-dataset>
29. NVIDIA Research Projects. *Flickr-Faces-HQ Dataset (FFHQ)* [en línia]. Actualitzat 12 desembre 2019. [Consulta: 10 maig 2021]. Disponible a: <https://github.com/NVlabs/ffhq-dataset>
30. Balraj Ashwath. *Stanford Background Dataset* [en línia]. Actualitzat novembre 2020. [Consulta: 1 juny 2021]. Disponible a: <https://www.kaggle.com/balraj98/stanford-background-dataset?select=images>
31. Lyudmil Vladimirov. *Training Custom Object Detector: Partition the Dataset* [en línia]. Actualitzat 26 agost 2020. [Consulta: 4 abril 2021]. Disponible a: <https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/training.html#partition-the-datase>
32. Girija Shankar Bejera. *TFRecords Explained* [en línia]. Publicat 15 octubre 2020. [Consulta: 4 juny 2021]. Disponible a: <https://towardsdatascience.com/tfrecords-explained-24b8f2133282>
33. Green Alliance. *A circular economy for smart devices* [en línia]. Publicat per Green Alliance, gener 2015. [Consulta: 10 juny 2021]. Disponible a: <https://green-alliance.org.uk/resources/A%20circular%20economy%20for%20smart%20devices.pdf>
34. Tecnología libre de conflicto. *The Environment and Electronic Devices* [en línia]. [Consulta: 10 juny 2021]. Disponible a: <https://www.tecnologialibredeconflicto.org/en/environment/>
35. NW Security Group Limited. *How Green is your IP Camera?* [en línia]. Publicat 27 maig 2017 per Network Webcams. [Consulta: 10 juny 2021]. Disponible a: <https://www.networkwebcams.co.uk/blog/2017/05/24/how-green-is-your-ip-camera>
36. Jobted. *Sueldo del Ingeniero en España* [en línia]. Actualitzat juny 2021. [Consulta: 10 juny 2021]. Disponible a: <https://www.jobted.es/salario/ingeniero>

Annex A: Codis

A1. Afegir text addicional al nom dels fitxers per evitar duplicitats

add_text_filename.py

```
1  from __future__ import print_function
2  from etaprogress.progress import ProgressBar
3  import argparse
4  import os
5  from os import listdir
6
7  # construct the argument parser and parse the arguments
8  ap = argparse.ArgumentParser()
9  ap.add_argument("-i", "--input_dir", required=True,
10     help="path to input dir")
11 ap.add_argument("-txt", "--text", required=True,
12     help="text to add")
13 ap.add_argument("-f", "--from", required=False,
14     help="from file")
15 ap.add_argument("-t", "--to", required=False,
16     help="to file")
17 args = vars(ap.parse_args())
18
19 files = listdir(args["input_dir"])
20 files.sort()
21
22 if args['to']:
23     files = files[:int(args['to'])]
24 if args['from']:
25     files = files[int(args['from']):]
26
27 bar = ProgressBar(len(files), max_width=100)
28
29 for i, file in enumerate(files):
30     bar.numerator = i + 1
```

```
31      print(bar, end='\r')
32
33      arr = file.split(".")
34      name = arr[0]
35      ext = arr[-1]
36      os.rename(os.path.join(args["input_dir"], file), os.path.join(args["input_dir"], name + args["text"] + "."
    + ext))
37
38  print()
```

A2. Afegir text addicional al *filename* de les anotacions

xml_add_text_filename.py

```
1   from __future__ import print_function
2   from etaprogress.progress import ProgressBar
3   import argparse
4   import os
5   from os import listdir
6   import xml.etree.ElementTree as ET
7
8   # construct the argument parser and parse the arguments
9   ap = argparse.ArgumentParser()
10  ap.add_argument("-i", "--input_dir", required=True,
11      help="path to input dir")
12  ap.add_argument("-f", "--from", required=False,
13      help="from file")
14  ap.add_argument("-t", "--to", required=False,
15      help="to file")
16  args = vars(ap.parse_args())
17
18  files = listdir(args["input_dir"])
19  files.sort()
20
21  PATH_TO_INPUT = args["input_dir"]
22
23  if args['to']:
24      files = files[:int(args['to'])]
```

```
25  if args['from']:
26      files = files[int(args['from']):]
27
28  bar = ProgressBar(len(files), max_width=100)
29
30  for i, file in enumerate(files):
31      bar.numerator = i + 1
32      print(bar, end='\r')
33
34      tree = ET.parse(os.path.join(PATH_TO_INPUT, file))
35      root = tree.getroot()
36
37      for cls in root.iter('filename'):
38          ext = cls.text.split(".")[-1]
39          name = file.split(".")[0]
40          cls.text = name + "." + ext
41
42      tree.write(os.path.join(PATH_TO_INPUT, file))
43  print()
```

A3. Canvi aleatori de brillantor en les imatges

brightness_da.py

```
1   # import the necessary packages
2   from __future__ import print_function
3   from etaprogress.progress import ProgressBar
4   import sys
5   import os
6   import cv2
7   import os.path
8   from os import listdir
9   import argparse
10  from functions import get_random_bright
11
12  # construct the argument parser and parse the arguments
13  ap = argparse.ArgumentParser()
```

```
ap.add_argument("-i", "--input_dir", required=True,
    help="path to input dir")
ap.add_argument("-o", "--output_dir", required=True,
    help="path to output dir")
ap.add_argument("-f", "--from", required=False,
    help="from file")
ap.add_argument("-t", "--to", required=False,
    help="to file")
args = vars(ap.parse_args())

PATH_TO_INPUT = args["input_dir"]
PATH_TO_OUTPUT = args["output_dir"]

files = listdir(PATH_TO_INPUT)
files.sort()

if args["to"]:
    files = files[:int(args["to"])]
if args["from"]:
    files = files[int(args["from"]):]

bar = ProgressBar(len(files), max_width=100)

for i, file in enumerate(files):
    bar.numerator = i + 1
    print(bar, end='\r')
    img = cv2.imread(os.path.join(PATH_TO_INPUT, file))

    img = get_random_bright(img)
    cv2.imwrite(os.path.join(PATH_TO_OUTPUT + file), img)

print()
```

functions.py

```
import cv2
import numpy as np
import random
```

```
4   import math
5
6   def rotate_image(image, angle):
7     image_center = tuple(np.array(image.shape[1::-1]) / 2)
8     rot_mat = cv2.getRotationMatrix2D(image_center, angle, 1.0)
9     result = cv2.warpAffine(image, rot_mat, image.shape[1::-1], flags=cv2.INTER_LINEAR)
10    return result
11
12  def get_random_bright(img):
13      fun = random.choice(functions)
14      output = fun(img)
15      return output
16
17  def whole_image(img):
18      rint = random.randint(50, 150)
19      bright = np.ones(img.shape, dtype="uint8") * rint
20      output = cv2.subtract(img, bright) if random.randint(0, 1) else cv2.add(img, bright)
21      return output
22
23  def get_bands(img):
24      n_cols = random.randint(1, 10)
25      shape = img.shape
26      brights = []
27      h, mod = divmod(shape[0], n_cols)
28      for i in range(0,n_cols):
29          op = h if not i + 1 == n_cols else h + mod
30          if random.randint(0, 1):
31              rint = random.randint(50, 150)
32              brights.append(np.array(np.ones((op,shape[1],shape[2]), dtype="uint8") * rint))
33          else:
34              brights.append(np.array(np.zeros((op,shape[1],shape[2]), dtype="uint8")))
35
36      bright = np.concatenate(tuple(brights))
37      return bright
38
39  def rows(img):
```

```
40      bright = get_bands(img)
41      output = cv2.subtract(img, bright) if random.randint(0, 1) else cv2.add(img, bright)
42      return output
43
44  def columns(img):
45      bright = get_bands(img)
46      bright = rotate_image(bright, 90)
47      output = cv2.subtract(img, bright) if random.randint(0, 1) else cv2.add(img, bright)
48      return output
49
50  def squares(img):
51      bright = get_bands(img)
52      bright = bright + rotate_image(bright, 90)
53      output = cv2.subtract(img, bright) if random.randint(0, 1) else cv2.add(img, bright)
54      return output
55
56  def diagonals(img):
57      bright = get_bands(img)
58      bright = rotate_image(bright, 45) if random.randint(0, 1) else rotate_image(bright, 135)
59      output = cv2.subtract(img, bright) if random.randint(0, 1) else cv2.add(img, bright)
60      return output
61
62  functions = [whole_image, rows, columns, squares, diagonals]
```

A4. Concatenar imatges horitzontalment

concat_images.py

```
1   # import the necessary packages
2   from __future__ import print_function
3   from etaprogress.progress import ProgressBar
4   import os
5   import cv2
6   import numpy as np
7   import os.path
8   from os import listdir
9   import argparse
10  import xml.etree.ElementTree as ET
```

```
import json
from datetime import datetime

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--input_dir", required=True,
    help="path to input dir")

args = vars(ap.parse_args())

PATH_TO_INPUT = args["input_dir"]

files = listdir(PATH_TO_INPUT)
files.sort()

i = 0
bar = ProgressBar(len(files), max_width=40)
imgs = []
for i, file in enumerate(files):
    bar.numerator = i + 1
    print(bar, end='\r')

    im = cv2.imread(os.path.join(PATH_TO_INPUT, file), cv2.IMREAD_UNCHANGED)
    width = int(im.shape[1]*480/im.shape[0])
    im = cv2.resize(im, (width, 480))
    imgs.append(im)

img = cv2.hconcat(imgs)
cv2.imwrite('output.png', img)
```

A5. Crear matriu de confusió a partir de model i imatges de *test*

confusion_matrix.py

```
# import the necessary packages
from __future__ import print_function
from etaprogress.progress import ProgressBar
import sys
```

```
import time
from object_detection.utils import label_map_util
from object_detection.utils import config_util
from object_detection.utils import visualization_utils as viz_utils
from object_detection.builders import model_builder
import tensorflow as tf
import os
import cv2
import numpy as np
import os.path
from os import listdir
import argparse
import xml.etree.ElementTree as ET
import json
from datetime import datetime

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--input_dir", required=True,
    help="path to input dir")
ap.add_argument("-cfg", "--config", required=True,
    help="path to model dir")
ap.add_argument("-l", "--label", required=True,
    help="path to label pbtxt")
ap.add_argument("-c", "--ckpt", required=True,
    help="path to checkpoint")
args = vars(ap.parse_args())

PATH_TO_INPUT = args["input_dir"]
PATH_TO_CFG = args["config"]
PATH_TO_LABEL = args["label"]
PATH_TO_CKPT = args["ckpt"]

JSON_NAME = datetime.today().strftime('%Y_%m_%d-%H_%M_%S')

cat_ind = ["mask", "no-mask"]
results = {
```

```
    "info":{
        "input_dir": PATH_TO_INPUT,
        "config": PATH_TO_CFG,
        "label": PATH_TO_LABEL,
        "ckpt": PATH_TO_CKPT
    },
    "confusion_matrix": {
        "right_mask": 0, "wrong_mask": 0, "mask_ratio": 0,
        "right_no-mask": 0, "wrong_no-mask": 0, "no-mask_ratio": 0
    }
}
# Load pipeline config and build a detection model
configs = config_util.get_configs_from_pipeline_file(PATH_TO_CFG)
model_config = configs['model']
detection_model = model_builder.build(model_config=model_config, is_training=False)

# Restore checkpoint
ckpt = tf.compat.v2.train.Checkpoint(model=detection_model)
ckpt.restore(PATH_TO_CKPT).expect_partial()

@tf.function
def detect_fn(image):
    """Detect objects in image."""

    image, shapes = detection_model.preprocess(image)
    prediction_dict = detection_model.predict(image, shapes)
    detections = detection_model.postprocess(prediction_dict, shapes)

    return detections

category_index = label_map_util.create_category_index_from_labelmap(PATH_TO_LABEL)

files = listdir(PATH_TO_INPUT)
files.sort()

i = 0
bar = ProgressBar(len(files[0::2]), max_width=40)
```

```
79  for img, xml in zip(files[0::2], files[1::2]):
80      bar.numerator = i + 1
81      print(bar, end='\r')
82      sys.stdout.flush()
83
84      frame = cv2.imread(os.path.join(PATH_TO_INPUT, img))
85
86      image_np = np.array(frame)
87
88      input_tensor = tf.convert_to_tensor(np.expand_dims(image_np, 0), dtype=tf.float32)
89      detections = detect_fn(input_tensor)
90
91      num_detections = int(detections.pop('num_detections'))
92      detections = {key: value[0, :num_detections].numpy()
93                    for key, value in detections.items()}
94      detections['num_detections'] = num_detections
95
96      # detection_classes should be ints.
97      detections['detection_classes'] = detections['detection_classes'].astype(np.int64)
98
99      tree = ET.parse(os.path.join(PATH_TO_INPUT, xml))
100     root = tree.getroot()
101
102     classes = []
103     for cls in root.iter('name'):
104         classes.append(cls.text)
105
106     found_classes = []
107
108     for cl, sc in zip(detections['detection_classes'], detections['detection_scores']):
109         if sc < 0.5:
110             break
111         found_classes.append(cat_ind[cl])
112
113     found_classes = found_classes[:len(classes)]
114     for cls in classes:
115         if cls in found_classes:
```

```
116             results["confusion_matrix"]["right_" + cls] += 1
117             found_classes.remove(cls)
118         else:
119             results["confusion_matrix"]["wrong_" + cls] += 1
120
121     with open('json/' + JSON_NAME + '.json', 'w') as json_file:
122         json.dump(results, json_file)
123     time.sleep(1)
124     i += 1
125
126 results["confusion_matrix"]["mask_ratio"] = results["confusion_matrix"]["right_mask"]
    /(results["confusion_matrix"]["right_mask"] + results["confusion_matrix"]["wrong_mask"])
127 results["confusion_matrix"]["no-mask_ratio"] = results["confusion_matrix"]["right_no-mask"]
    /(results["confusion_matrix"]["right_no-mask"] + results["confusion_matrix"]["wrong_no-mask"])
128
129 print(results)
130 with open('json/' + JSON_NAME + '.json', 'w') as json_file:
131     json.dump(results, json_file)
```

A6. Retallar imatges en massa

crop_images.py

```
1  import argparse
2  from os import listdir
3  import cv2
4
5  # construct the argument parser and parse the arguments
6  ap = argparse.ArgumentParser()
7  ap.add_argument("-i", "--input_dir", required=True,
8      help="path to input dir")
9  ap.add_argument("-o", "--output_dir", required=True,
10     help="path to input dir")
11 ap.add_argument("-he", "--height", required=True,
12     help="path to input dir")
13 ap.add_argument("-w", "--width", required=True,
14     help="path to input dir")
```

```
ap.add_argument("-ext", "--extension", required=False,
    help="orgin file by default")
args = vars(ap.parse_args())

if (args["extension"]):
    ext = args["extension"]

files = listdir(args["input_dir"])
files.sort()

for i, file in enumerate(files):
    if not ext:
        ext_aux = file.split(".")[-1]
    else :
        ext_aux = ext

    img = cv2.imread(args["input_dir"] + file)
    shape = img.shape
    h = int(args["height"])
    w = int(args["width"])

    y = round((shape[0]-int(h))/2)
    x = round((shape[1]-int(w))/2)

    crop_img = img[y:y+h, x:x+w]
    cv2.imwrite(args["output_dir"] + file.split(".")[0] + "." + ext, crop_img)
    cv2.waitKey(0)
```

A7. Informació de les extensions trobades dins d'una carpeta

files_ext_info.py

```
from os import listdir
import argparse


files = listdir("/media/ignasi/TOSHIBA EXT/TFG/datasets/mixed/train")
files.sort()
```

```
6
7   exts = {}
8   unmatched = {}
9   for i, file in enumerate(files):
10      info = file.split(".")
11      name = info[0]
12      ext = info[-1]
13      if ext in exts:
14          exts[ext] += 1
15      else:
16          exts[ext] = 1
17
18  print(exts)
```

A8. Filtrar i esborrar aquelles imatges en les quals no s'ha detectat cares

filter_undetected.py

```
1   from __future__ import print_function
2   from etaprogress.progress import ProgressBar
3   import argparse
4   import os
5   from os import listdir
6
7   # construct the argument parser and parse the arguments
8   ap = argparse.ArgumentParser()
9   ap.add_argument("-m", "--masked_dir", required=True,
10     help="path to masked dir")
11  ap.add_argument("-i", "--input_dir", required=True,
12     help="path to input dir")
13  ap.add_argument("-f", "--filter_dir", required=False,
14     help="path to filter dir")
15  args = vars(ap.parse_args())
16
17  masks = listdir(args["masked_dir"])
18  masks.sort()
19
20  bar = ProgressBar(len(masks), max_width=100)
```

```
21  detected = []
22  for i, mask in enumerate(masks):
23      bar.numerator = i + 1
24      print(bar, end='\r')
25      detect = mask.split("_")[0]
26      if not (detect in detected):
27          detected.append(detect)
28
29  files = listdir(args["input_dir"])
30  files.sort()
31  bar = ProgressBar(len(files), max_width=100)
32  for i, file in enumerate(files):
33      bar.numerator = i + 1
34      print(bar, end='\r')
35      if os.path.isdir(args["input_dir"] + file):
36          continue
37
38      if not (file.split(".")[0] in detected):
39          os.remove(args["input_dir"] + file)
40          #os.rename(args["input_dir"] + file, args["filter_dir"] + file)
41  print()
```

A9. Generar cares amb mascareta

generate_masks.py

```
1   # import the necessary packages
2   from __future__ import print_function
3   from functions import rotate_image, get_masks, get_face, xml_masked_face, xml_mask, xml_face
4   import numpy as np
5   import argparse
6   import imutils
7   import dlib
8   import cv2
9   import os
10  import os.path
11  from os import listdir
12  import random
```

```
13  import sys
14  import time
15  from etaprogress.progress import ProgressBar
16
17  # construct the argument parser and parse the arguments
18  ap = argparse.ArgumentParser()
19  ap.add_argument("-p", "--shape-predictor", required=True,
20      help="path to facial landmark predictor")
21  ap.add_argument("-i", "--input_dir", required=True,
22      help="path to input dir")
23  ap.add_argument("-m", "--mask_dir", required=True,
24      help="path to mask dir")
25  ap.add_argument("-o", "--output_dir", required=True,
26      help="path to output dir")
27  ap.add_argument("-oa", "--output_ann", required=True,
28      help="path to output annotations dir")
29  ap.add_argument("-s", "--show", required=False,
30      help="show results (False by defalut)")
31  ap.add_argument("-c", "--ckpt", required=False,
32      help="checkpoint")
33  args = vars(ap.parse_args())
34
35  # initialize dlib's face detector (HOG-based) and then create
36  # the facial landmark predictor
37  detector = dlib.get_frontal_face_detector()
38  predictor = dlib.shape_predictor(args["shape_predictor"])
39
40  # get masks
41  masks = get_masks("/home/ignasi/Documents/TFG/datasets/alpha_masks/prod/")
42
43  files = listdir(args["input_dir"])
44  files.sort()
45
46  if args["ckpt"]:
47      files = files[int(args["ckpt"]):]
48
49  bar = ProgressBar(len(files), max_width=100)
```

```
50
51  for i, file in enumerate(files):
52      bar.numerator = i + 1
53      print(bar, end='\r')
54      sys.stdout.flush()
55
56      # load the input image and convert it to grayscale
57      img1 = cv2.imread(args["input_dir"] + file)
58      gray = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
59
60      # detect faces in the grayscale image
61      rects = detector(gray, 1)
62
63      if len(rects) < 1:
64          continue
65
66      face = get_face(rects, predictor, gray)
67      xml_face(args["input_dir"], args["output_ann"], file.split(".")[0], face, img1.shape)
68
69      for type in masks[face["position"]].keys():
70          img_out = img1
71          #take one random mask for each type
72          img2 = random.choice(masks[face["position"]][type])
73
74          # resize and rotate
75          img2 = cv2.resize(img2, (face["width"], face["height"]))
76          img2 = rotate_image(img2, face["angle"])
77
78          # to numpy arrays
79          img_out = np.array(img_out)
80          img2 = np.array(img2)
81
82          # all pixels that are going to be replaced
83          for (y2, y1) in enumerate(range(0 + face["y_offset"], face["height"] + face["y_offset"])):
84              for (x2, x1) in enumerate(range(0 + face["x_offset"], face["width"] + face["x_offset"])):
85
86                  # if alpha = 0 old pixel remains
```

```
87                alpha = round(img2[y2][x2][3]/255)
88                img_out[y1][x1] = img_out[y1][x1] * (1-alpha) + img2[y2][x2][:3]*alpha
89
90          out_filename = file.split(".")[0] + "_" + face["position"] + "_" + type
91          xml_masked_face(args["output_dir"], args["output_ann"], out_filename, face, img_out.shape)
92          xml_mask(args["output_dir"], args["output_ann"], out_filename, face, img_out.shape, type)
93
94          # write the output image with the mask
95          cv2.imwrite(args["output_dir"] + out_filename + ".png", img_out)
96
97          if args["show"]:
98              cv2.imshow("Output", img_out)
99              cv2.waitKey(0)
100 print()
```

functions.py

```
1   import numpy as np
2   import cv2
3   from os import listdir
4   from imutils import face_utils
5   import math
6   import xml.etree.ElementTree as ET
7
8   def rotate_image(image, angle):
9     image_center = tuple(np.array(image.shape[1::-1]) / 2)
10    rot_mat = cv2.getRotationMatrix2D(image_center, angle, 1.0)
11    result = cv2.warpAffine(image, rot_mat, image.shape[1::-1], flags=cv2.INTER_LINEAR)
12    return result
13
14  def get_masks(mask_path):
15      mask_dir = listdir(mask_path)
16      ret = {}
17
18      for position in mask_dir:
19          pos_dir = listdir(mask_path + position)
20
21          for type in pos_dir:
```

```
22            masks = listdir(mask_path + position + "/" + type)
23
24            for mask in masks:
25                img = cv2.imread( mask_path + position + "/" + type + "/" + mask,
    cv2.IMREAD_UNCHANGED)
26                if not position in ret:
27                    ret[position] = {type: [img]}
28                else:
29                    if not type in ret[position]:
30                        ret[position][type] = [img]
31                    else:
32                        ret[position][type].append(img)
33    return ret
34
35 def get_face(rects, predictor, gray):
36
37    max_area = 0
38    for rect in rects:
39        # calc area
40        area = (rect.right() - rect.left()) * (rect.bottom() - rect.top())
41
42        # make sure to get biggest face
43        if area < max_area:
44            continue
45
46        max_area = area
47
48        # determine the facial landmarks for the face region, then
49        # convert the facial landmark (x, y)-coordinates to a NumPy array
50        shape = predictor(gray, rect)
51        shape = face_utils.shape_to_np(shape)
52
53        edge = gray.shape[0]
54
55        left_ear = parse_coords(shape[1], edge)
56        chin = parse_coords(shape[8], edge)
57        right_ear = parse_coords(shape[15], edge)
```

```
58          nose = parse_coords(shape[29], edge)
59
60          x_offset = left_ear[0]
61          y_offset = nose[1]
62
63          width = right_ear[0] - left_ear[0]
64          height = chin[1] - nose[1]
65
66          angle = (right_ear[1] - left_ear[1])/width
67          angle = -int(round(math.atan(angle)*180/math.pi,0))
68
69          topleft = (parse_coord(shape[2][0], edge),parse_coord(y_offset - height, edge))
70          botright = (parse_coord(shape[14][0], edge),parse_coord(shape[9][1], edge))
71
72          left_distance = nose[0] - left_ear[0]
73          right_distance = right_ear[0] - nose[0]
74
75          ratio = left_distance/right_distance
76          position = "c"
77          if ratio > 4:
78              position = "l"
79          elif ratio > 1.5:
80              position = "lc"
81          elif ratio < 0.25:
82              position = "r"
83          elif ratio < 2/3:
84              position = "rc"
85
86          face = {
87              "position": position,
88              "x_offset": x_offset,
89              "y_offset": y_offset,
90              "width": width,
91              "height": height,
92              "angle": angle,
93              "topleft": topleft,
94              "botright": botright
```

```
95          }
96
97      return face
98
99  def xml_masked_face(imgs_path, ann_path, file, face, shape):
100     tree = ET.parse('xml/dist.xml')
101     root = tree.getroot()
102
103     root.find("folder").text = imgs_path.split("/")[-2]
104     root.find("filename").text = file + ".png"
105     root.find("path").text = imgs_path
106
107     root.find("size").find("height").text = str(shape[0])
108     root.find("size").find("width").text = str(shape[1])
109     root.find("size").find("depth").text = str(shape[2])
110
111     root.find("object").find("name").text = "mask"
112     root.find("object").find("bndbox").find("xmin").text = str(face["topleft"][0])
113     root.find("object").find("bndbox").find("ymin").text = str(face["topleft"][1])
114     root.find("object").find("bndbox").find("xmax").text = str(face["botright"][0])
115     root.find("object").find("bndbox").find("ymax").text = str(face["botright"][1])
116
117     tree.write(ann_path + "masked_face/" + file + ".xml")
118
119 def xml_mask(imgs_path, ann_path, file, face, shape, type):
120     tree = ET.parse('xml/dist.xml')
121     root = tree.getroot()
122
123     root.find("folder").text = imgs_path.split("/")[-2]
124     root.find("filename").text = file + ".png"
125     root.find("path").text = imgs_path
126
127     root.find("size").find("height").text = str(shape[0])
128     root.find("size").find("width").text = str(shape[1])
129     root.find("size").find("depth").text = str(shape[2])
130
131     root.find("object").find("name").text = type
```

```
132     root.find("object").find("bndbox").find("xmin").text = str(face["x_offset"])
133     root.find("object").find("bndbox").find("ymin").text = str(face["y_offset"])
134     root.find("object").find("bndbox").find("xmax").text = str(face["x_offset"] + face["width"])
135     root.find("object").find("bndbox").find("ymax").text = str(face["y_offset"] + face["height"])
136 
137     tree.write(ann_path + "mask/" + file + ".xml")
138 
139 def xml_face(imgs_path, ann_path, file, face, shape):
140     tree = ET.parse('xml/dist.xml')
141     root = tree.getroot()
142 
143     root.find("folder").text = imgs_path.split("/")[-2]
144     root.find("filename").text = file + ".png"
145     root.find("path").text = imgs_path
146 
147     root.find("size").find("height").text = str(shape[0])
148     root.find("size").find("width").text = str(shape[1])
149     root.find("size").find("depth").text = str(shape[2])
150 
151     root.find("object").find("name").text = "no-mask"
152     root.find("object").find("bndbox").find("xmin").text = str(face["topleft"][0])
153     root.find("object").find("bndbox").find("ymin").text = str(face["topleft"][1])
154     root.find("object").find("bndbox").find("xmax").text = str(face["botright"][0])
155     root.find("object").find("bndbox").find("ymax").text = str(face["botright"][1])
156 
157     tree.write(ann_path + "face/" + file + ".xml")
158 
159 def parse_coords(coords, edge):
160 
161     for i, coord in enumerate(coords):
162 
163         coords[i] = parse_coord(coord, edge)
164 
165     return coords
166 
167 def parse_coord(coord, edge):
168 
```

```
169     if not (0 <= coord <= edge):
170         if coord < 0:
171             coord = 0
172         else:
173             coord = edge
174 
175     return coord
```

A10. Redimensionar imatges en massa

image_resizer.py

```
1   from __future__ import print_function
2   from etaprogress.progress import ProgressBar
3   import argparse
4   import cv2
5   from os import listdir
6   
7   ap = argparse.ArgumentParser()
8   ap.add_argument("-i", "--input", required=True,
9       help="path to input dir")
10  ap.add_argument("-o", "--output", required=True,
11      help="path to output dir")
12  ap.add_argument("-w", "--width", required=True,
13      help="output width")
14  ap.add_argument("-he", "--height", required=True,
15      help="output height")
16  args = vars(ap.parse_args())
17  
18  i_dir = args["input"]
19  o_dir = args["output"]
20  width = args["width"]
21  height = args["height"]
22  
23  files = listdir(i_dir)
24  files.sort()
25  
26  bar = ProgressBar(len(files), max_width=100)
```

```
27
28  for i, file in enumerate(files):
29      bar.numerator = i + 1
30      print(bar, end='\r')
31
32      img = cv2.imread(i_dir + file)
33      shape = img.shape
34
35      if shape[0] == height and shape[1] == width:
36          continue
37
38      img = cv2.resize(img, (int(width), int(height)))
39
40      cv2.imwrite(o_dir + file, img)
41  print()
```

A11. Objectes etiquetats, redimensionats i col·locats aleatòriament sobre un fons aleatori

labeled_to_background.py

```
1   # import the necessary packages
2   from __future__ import print_function
3   from etaprogress.progress import ProgressBar
4   import sys
5   import time
6   import os
7   import cv2
8   import numpy as np
9   import os.path
10  from os import listdir
11  import argparse
12  import xml.etree.ElementTree as ET
13  import random
14
15  # construct the argument parser and parse the arguments
16  ap = argparse.ArgumentParser()
```

```
17  ap.add_argument("-i", "--input_dir", required=True,
18      help="path to input dir")
19  ap.add_argument("-o", "--output_dir", required=True,
20      help="path to output dir")
21  ap.add_argument("-ix", "--input_xml", required=True,
22      help="path to input xml dir")
23  ap.add_argument("-ox", "--output_xml", required=True,
24      help="path to outpùt xml dir")
25  ap.add_argument("-b", "--background_dir", required=True,
26      help="path to background dir")
27  ap.add_argument("-f", "--from", required=False,
28      help="from file")
29  ap.add_argument("-t", "--to", required=False,
30      help="to file")
31  args = vars(ap.parse_args())
32
33  PATH_TO_INPUT = args["input_dir"]
34  PATH_TO_OUTPUT = args["output_dir"]
35  PATH_TO_INPUT_XML = args["input_xml"]
36  PATH_TO_OUTPUT_XML = args["output_xml"]
37  PATH_TO_BACKGROUND = args["background_dir"]
38
39  files = listdir(PATH_TO_INPUT)
40  files.sort()
41
42  bgs = listdir(PATH_TO_BACKGROUND)
43  bgs.sort()
44
45  if args["to"]:
46      files = files[:int(args["to"])]
47  if args["from"]:
48      files = files[int(args["from"]):]
49
50  bar = ProgressBar(len(files), max_width=100)
51
52  for i, file in enumerate(files):
53      bg = cv2.imread(os.path.join(PATH_TO_BACKGROUND, random.choice(bgs)))
```

```
54      face = cv2.imread(os.path.join(PATH_TO_INPUT, file))
55      tree = ET.parse(os.path.join(PATH_TO_INPUT_XML, file.split(".")[0] + ".xml"))
56      root = tree.getroot()
57
58      bg_shape = bg.shape
59      for bb in root.iter('bndbox'):
60          xmin = int(bb.find('xmin').text)
61          ymin = int(bb.find('ymin').text)
62          xmax = int(bb.find('xmax').text)
63          ymax = int(bb.find('ymax').text)
64          hw = [ymax - ymin, xmax - xmin]  ##original face dimensions
65
66          # new face height (20-80% of background height)
67          nh = random.randint(int(0.2*bg_shape[0]), int(0.8*bg_shape[0]))
68          # new face width (60-100% proportioned respect height)
69          nw = int(random.uniform(0.6,1)*hw[1]*nh/hw[0])
70
71          ny = random.randint(0, bg_shape[0] - nh) # random y
72          nx = random.randint(0, bg_shape[1] - nw) # random x
73
74          # get face from whole original image
75          roi_face = face[ymin:ymax, xmin:xmax, :]
76
77          # resize face to calcd dimensions
78          roi_face = cv2.resize(roi_face, (nw, nh))
79          # subsitute new face in calcd position
80          bg[ny:(ny+nh), nx:(nx+nw), :] = roi_face
81
82          bb.find('xmin').text = str(nx)
83          bb.find('ymin').text = str(ny)
84          bb.find('xmax').text = str(nx + nw)
85          bb.find('ymax').text = str(ny + nh)
86
87      cv2.imwrite(os.path.join(PATH_TO_OUTPUT + file), bg)
88      tree.write(os.path.join(PATH_TO_OUTPUT_XML, file.split(".")[0] + ".xml"))
89
90      bar.numerator = i + 1
```

```
91      print(bar, end='\r')
92  print()
```

A12. Canviar categoria antiga per una nova en l'anotació de les imatges

change_cat_xml.py

```
1   # import the necessary packages
2   from __future__ import print_function
3   from etaprogress.progress import ProgressBar
4   import sys
5   import time
6   import os
7   import cv2
8   import numpy as np
9   import os.path
10  from os import listdir
11  import argparse
12  import xml.etree.ElementTree as ET
13  import json
14  from datetime import datetime
15
16  # construct the argument parser and parse the arguments
17  ap = argparse.ArgumentParser()
18  ap.add_argument("-i", "--input_dir", required=True,
19      help="path to input dir")
20  ap.add_argument("-o", "--output_dir", required=True,
21      help="path to output dir")
22  ap.add_argument("-c", "--category", required=True,
23      help="new category")
24  ap.add_argument("-ctg", "--category_to_change", required=True,
25      help="old categroy")
26  args = vars(ap.parse_args())
27
28  cat = args['category']
29
30  PATH_TO_INPUT = args["input_dir"]
31  PATH_TO_OUPUT = args["output_dir"]
```

```
32
33  files = listdir(PATH_TO_INPUT)
34  files.sort()
35
36  bar = ProgressBar(len(files), max_width=100)
37  for i, file in enumerate(files):
38      bar.numerator = i + 1
39      print(bar, end='\r')
40
41      tree = ET.parse(os.path.join(PATH_TO_INPUT, file))
42      root = tree.getroot()
43
44      for cls in root.iter('name'):
45          if cls.text == args["category_to_change"]:
46              cls.text = cat
47
48      tree.write(os.path.join(PATH_TO_OUPUT, file))
49  print()
```

A13. Unió de TFRecords

merge_tfrecord.py

```
1   # import the necessary packages
2   from __future__ import print_function
3   from etaprogress.progress import ProgressBar
4   import sys
5   import time
6   from object_detection.utils import label_map_util
7   from object_detection.utils import config_util
8   from object_detection.utils import visualization_utils as viz_utils
9   from object_detection.builders import model_builder
10  import tensorflow as tf
11  import os
12  import cv2
13  import numpy as np
14  import os.path
15  from os import listdir
```

```
16  import argparse
17  import xml.etree.ElementTree as ET
18  import json
19  from datetime import datetime
20
21  # construct the argument parser and parse the arguments
22  ap = argparse.ArgumentParser()
23  ap.add_argument("-i", "--input_dir", required=True,
24      help="path to input dir")
25  ap.add_argument("-o", "--output_file", required=True,
26      help="path to output file")
27  args = vars(ap.parse_args())
28
29  PATH_TO_INPUT = args["input_dir"]
30  PATH_TO_OUTPUT = args["output_file"]
31
32  files = listdir(PATH_TO_INPUT)
33  files.sort()
34
35  list_of_tfrecord_files = []
36  for i, file in enumerate(files):
37      if file.split(".")[-1] == 'record':
38          list_of_tfrecord_files.append(os.path.join(PATH_TO_INPUT,file))
39
40  dataset = tf.data.TFRecordDataset(list_of_tfrecord_files)
41
42  # Save dataset to .tfrecord file
43  filename = 'result.record'
44  writer = tf.data.experimental.TFRecordWriter(PATH_TO_OUTPUT)
45  writer.write(dataset)
```

A14. Moure o copiar fitxers en massa. Possibilitat de filtrar per posició i extensió del fitxer

move_files.py

```
1  from __future__ import print_function
2  from etaprogress.progress import ProgressBar
3  from shutil import copyfile
4  import argparse
5  import os
6  from os import listdir
7
8  # construct the argument parser and parse the arguments
9  ap = argparse.ArgumentParser()
10 ap.add_argument("-o", "--output_dir", required=True,
11    help="path to output dir")
12 ap.add_argument("-i", "--input_dir", required=True,
13    help="path to input dir")
14 ap.add_argument("-f", "--from", required=False,
15    help="from file")
16 ap.add_argument("-t", "--to", required=False,
17    help="to file")
18 ap.add_argument("-c", "--copy", required=False,
19    help="copy, not move")
20 ap.add_argument("-e", "--extension", required=False,
21    help="only move files with this extension")
22 args = vars(ap.parse_args())
23
24 copy = True if args['copy'] else False
25
26 files = listdir(args["input_dir"])
27 files.sort()
28
29 fr = 0
30 if args['from']:
31    fr = int(args['from'])
32
33 to = len(files)
34 if args['to']:
35    to = int(args['to'])
36
37 ext = args["extension"] if args["extension"] else False
```

```
38
39  bar = ProgressBar(to - fr, max_width=100)
40
41  for i in range(fr, to):
42      file = files[i]
43      if ext:
44          if not ext == file.split(".")[-1]:
45              continue
46      bar.numerator = i + 1 -fr
47      print(bar, end='\r')
48      if os.path.isdir(args["input_dir"] + file):
49          continue
50      if copy:
51          copyfile(args["input_dir"] + file, args["output_dir"] + file)
52          continue
53      os.rename(args["input_dir"] + file, args["output_dir"] + file)
54  print()
```

A15. Llistar i comptar diferents classes d'objectes en anotació JSON

classes_count.py

```
1   import json
2   import os
3   from os import listdir
4
5   dir = "annotations/"
6
7   files = listdir(dir)
8
9   classes = {'differentClasses': 0, 'classNames': {}}
10
11  for file in files:
12      with open(dir + file) as f:
13          fs = json.load(f)
14          anns = fs["Annotations"]
15          for ann in anns:
16              className = ann["classname"]
```

```
            if className in classes["classNames"]:
                classes["classNames"][className] += 1
            else:
                classes["differentClasses"] += 1
                classes["classNames"][className] = 1

print(classes)
with open('classes.json', 'w') as json_file:
  json.dump(classes, json_file)
```

A16. Anotació JSON a XML

json_to_xml.py

```
import cv2
import json
import os
import os.path
from os import listdir
import xml.etree.ElementTree as ET

dir = “annotations/"
img_dir = "json_images/"
img_dir_name = "json_images"
output_dir = "json_to_xml/"
xml_dir = "../xml/"

coi = {
    "face_no_mask": "face_no_mask",
    "face_with_mask": "face_with_mask",
    "face_with_mask_incorrect": "face_no_mask",
    "face_other_covering": "face_no_mask"
}

files = listdir(dir)
files.sort()

tree = ET.parse(xml_dir + 'dist.xml')
root = tree.getroot()
```

```
26
27  root.find("folder").text = img_dir_name
28  root.find("path").text = img_dir
29
30  tree.write(xml_dir + 'tmp.xml')
31
32  for idx, file in enumerate(files):
33      print(idx, file)
34
35      file_wo_ext = file.split(".")[0]
36      file_wo_json = file.jso")[0]
37      tree = ET.parse(xml_dir + 'tmp.xml')
38      root = tree.getroot()
39      root.find("path").text = img_dir + file_wo_json
40
41      with open(dir + file) as f:
42          fs = json.load(f)
43          filename = fs["FileName"]
44          if not os.path.isfile(img_dir + filename):
45              continue
46
47          root.find("filename").text = filename
48
49          img = cv2.imread(img_dir + filename)
50          shape = img.shape
51
52          root.find("size").find("height").text = str(shape[0])
53          root.find("size").find("width").text = str(shape[1])
54          root.find("size").find("depth").text = str(shape[2])
55
56          anns = fs["Annotations"]
57          for ann in anns:
58              className = ann["classname"]
59
60              if className in coi:
61                  obj = ET.Element("object")
```

```
62
63              name = ET.SubElement(obj, "name")
64              pose = ET.SubElement(obj, "pose")
65              truncated = ET.SubElement(obj, "truncated")
66              difficult = ET.SubElement(obj, "difficult")
67              bndbox = ET.SubElement(obj, "bndbox")
68              name.text = coi[className]
69              pose.text = "Unspecified"
70              truncated.text = "0"
71              difficult.text = "0"
72
73              xmin = ET.SubElement(bndbox, "xmin")
74              ymin = ET.SubElement(bndbox, "ymin")
75              xmax = ET.SubElement(bndbox, "xmax")
76              ymax = ET.SubElement(bndbox, "ymax")
77
78              boundingBox = ann["BoundingBox"]
79              xmin.text = str(boundingBox[0])
80              ymin.text = str(boundingBox[1])
81              xmax.text = str(boundingBox[2])
82              ymax.text = str(boundingBox[3])
83
84              root.append(obj)
85
86          tree.write(output_dir + filename.split(".")[0] + '.xml')
87
88  os.remove(xml_dir + 'tmp.xml')
```

A17. Detectar cares amb Viola Jones i anotar-les

haar_to_xml.py

```
1  import cv2
2  import os
3  from os import listdir
4  import xml.etree.ElementTree as ET
5  import numpy as np
6
```

```
7   dir = "00000/"
8   dir_name = "00000"
9   output_dir = "00000_xml/"
10  xml_dir = "../xml/"
11
12  faceCascade=
    cv2.CascadeClassifier("/opt/opencv/data/haarcascades/haarcascade_frontalface_default.xml")
13  files = listdir(dir)
14
15  tree = ET.parse(xml_dir + 'dist.xml')
16  root = tree.getroot()
17
18  root.find("folder").text = dir_name
19  root.find("path").text = dir
20
21  tree.write(xml_dir + 'tmp.xml')
22
23  for idx, file in enumerate(files):
24      file_wo_ext = file.split(".")[0]
25      print(idx, file)
26
27      tree = ET.parse(xml_dir + 'tmp.xml')
28      root = tree.getroot()
29      root.find("filename").text = file
30      root.find("path").text = dir + file
31
32      img = cv2.imread(dir + file)
33      shape = img.shape
34
35      root.find("size").find("height").text = str(shape[0])
36      root.find("size").find("width").text = str(shape[1])
37      root.find("size").find("depth").text = str(shape[2])
38
39      imgGray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
40
41      faces = faceCascade.detectMultiScale(imgGray,1.1,4)
```

```
42
43      for (x,y,w,h) in faces:
44          cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
45          obj = ET.Element("object")
46
47          name = ET.SubElement(obj, "name")
48          pose = ET.SubElement(obj, "pose")
49          truncated = ET.SubElement(obj, "truncated")
50          difficult = ET.SubElement(obj, "difficult")
51          bndbox = ET.SubElement(obj, "bndbox")
52          name.text = "mask"
53          pose.text = "Unspecified"
54          truncated.text = "0"
55          difficult.text = "0"
56
57          xmin = ET.SubElement(bndbox, "xmin")
58          ymin = ET.SubElement(bndbox, "ymin")
59          xmax = ET.SubElement(bndbox, "xmax")
60          ymax = ET.SubElement(bndbox, "ymax")
61
62          xmin.text = str(x)
63          ymin.text = str(y)
64          xmax.text = str(x+w)
65          ymax.text = str(y+h)
66
67          root.append(obj)
68
69      tree.write(output_dir + file_wo_ext + '.xml')
70
71      """ cv2.imshow("Result", img)
72      cv2.waitKey(0)
73      cv2.imwrite("output/" + file, img) """
74      #break
75  os.remove(xml_dir + 'tmp.xml')
```

A18. Estructura anotació en XML

dist.xml

```
<annotation>
    <folder></folder>
    <filename></filename>
    <path></path>
    <source>
        <database>Unknown</database>
    </source>
    <size>
        <width></width>
        <height></height>
        <depth></depth>
    </size>
    <segmented>0</segmented>
</annotation>
```

A19. Anomenar fitxers amb notació incremental i amb possibilitat d'especificar dígits utilitzats i nombre en què començar

rename_files.py

```
import argparse
import os
from os import listdir

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--input_dir", required=True,
    help="path to input dir")
ap.add_argument("-d", "--digits", required=False,
    help="naming digits (4 = 0000)")
ap.add_argument("-f", "--from", required=False,
    help="from digit")
args = vars(ap.parse_args())

```

```
files = listdir(args["input_dir"])
files.sort()

fr = 0
if args["from"]:
    fr = int(args["from"])

digits = len(str(len(files) + fr))
if args["digits"]:
    digits = int(args["digits"])

for i, file in enumerate(files):
    name = str(i + fr)
    name = (digits - len(name)) * "0" + name + "." + file.split(".")[1]

    os.rename(args["input_dir"] + file, args["input_dir"] + name)
    print(i)
```

A20. Provar model amb imatges

test_model_view_images.py

```
# import the necessary packages
from __future__ import print_function
from etaprogress.progress import ProgressBar
import time
from object_detection.utils import label_map_util
from object_detection.utils import config_util
from object_detection.utils import visualization_utils as viz_utils
from object_detection.builders import model_builder
import tensorflow as tf
import os
import cv2
import numpy as np
import os.path
from os import listdir
import argparse

```

```
# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--input_dir", required=True,
    help="path to input dir")
ap.add_argument("-cfg", "--config", required=True,
    help="path to model dir")
ap.add_argument("-l", "--label", required=True,
    help="path to label pbtxt")
ap.add_argument("-c", "--ckpt", required=True,
    help="path to checkpoint")
ap.add_argument("-f", "--from", required=False,
    help="from which file")
ap.add_argument("-t", "--to", required=False,
    help="to which file")
args = vars(ap.parse_args())

PATH_TO_CFG = args["config"]
PATH_TO_LABEL = args["label"]
PATH_TO_CKPT = args["ckpt"]
PATH_TO_INPUT = args["input_dir"]

# Load pipeline config and build a detection model
configs = config_util.get_configs_from_pipeline_file(PATH_TO_CFG)
model_config = configs['model']
detection_model = model_builder.build(model_config=model_config, is_training=False)

# Restore checkpoint
ckpt = tf.compat.v2.train.Checkpoint(model=detection_model)
ckpt.restore(PATH_TO_CKPT).expect_partial()

@tf.function
def detect_fn(image):
    """Detect objects in image."""

    image, shapes = detection_model.preprocess(image)
    prediction_dict = detection_model.predict(image, shapes)
    detections = detection_model.postprocess(prediction_dict, shapes)
```

```
54
55      return detections
56
57  category_index = label_map_util.create_category_index_from_labelmap(PATH_TO_LABEL)
58
59  files = listdir(PATH_TO_INPUT)
60  files.sort()
61
62  if args["to"]:
63      files = files[:int(args["to"])]
64  if args["from"]:
65      files = files[int(args["from"]):]
66
67  for file in files:
68      if not file.split(".")[-1] == "png":
69          continue
70      frame = cv2.imread(PATH_TO_INPUT + file)
71      image_np = np.array(frame)
72
73      input_tensor = tf.convert_to_tensor(np.expand_dims(image_np, 0), dtype=tf.float32)
74      detections = detect_fn(input_tensor)
75
76      num_detections = int(detections.pop('num_detections'))
77      detections = {key: value[0, :num_detections].numpy()
78                    for key, value in detections.items()}
79      detections['num_detections'] = num_detections
80
81      # detection_classes should be ints.
82      detections['detection_classes'] = detections['detection_classes'].astype(np.int64)
83
84      label_id_offset = 1
85      image_np_with_detections = image_np.copy()
86
87      viz_utils.visualize_boxes_and_labels_on_image_array(
88              image_np_with_detections,
89              detections['detection_boxes'],
90              detections['detection_classes']+label_id_offset,
```

```
91                detections['detection_scores'],
92                category_index,
93                use_normalized_coordinates=True,
94                max_boxes_to_draw=5,
95                min_score_thresh=.5,
96                agnostic_mode=False)
97
98        cv2.imwrite('output.png', image_np_with_detections)
99
100       input("Press key to continue...")
```

A21. Provar model per webcam

test_model_webcam.py

```
1   # import the necessary packages
2   from __future__ import print_function
3   from etaprogress.progress import ProgressBar
4   import time
5   from object_detection.utils import label_map_util
6   from object_detection.utils import config_util
7   from object_detection.utils import visualization_utils as viz_utils
8   from object_detection.builders import model_builder
9   import tensorflow as tf
10  import os
11  import cv2
12  import numpy as np
13  import os.path
14  from os import listdir
15  import argparse
16
17  # construct the argument parser and parse the arguments
18  ap = argparse.ArgumentParser()
19  ap.add_argument("-cfg", "--config", required=True,
20      help="path to model dir")
21  ap.add_argument("-l", "--label", required=True,
22      help="path to label pbtxt")
23  ap.add_argument("-c", "--ckpt", required=True,
```

```
    help="path to checkpoint")
args = vars(ap.parse_args())

PATH_TO_CFG = args["config"]
PATH_TO_LABEL = args["label"]
PATH_TO_CKPT = args["ckpt"]

# Load pipeline config and build a detection model
configs = config_util.get_configs_from_pipeline_file(PATH_TO_CFG)
model_config = configs['model']
detection_model = model_builder.build(model_config=model_config, is_training=False)

# Restore checkpoint
ckpt = tf.compat.v2.train.Checkpoint(model=detection_model)
ckpt.restore(PATH_TO_CKPT).expect_partial()

@tf.function
def detect_fn(image):
    """Detect objects in image."""

    image, shapes = detection_model.preprocess(image)
    prediction_dict = detection_model.predict(image, shapes)
    detections = detection_model.postprocess(prediction_dict, shapes)

    return detections

category_index = label_map_util.create_category_index_from_labelmap(PATH_TO_LABEL)

# Setup capture
cap = cv2.VideoCapture(0)
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

while True:
    ret, frame = cap.read()
    image_np = np.array(frame)

```

```
61      input_tensor = tf.convert_to_tensor(np.expand_dims(image_np, 0), dtype=tf.float32)
62      detections = detect_fn(input_tensor)
63
64      num_detections = int(detections.pop('num_detections'))
65      detections = {key: value[0, :num_detections].numpy()
66                     for key, value in detections.items()}
67      detections['num_detections'] = num_detections
68
69      # detection_classes should be ints.
70      detections['detection_classes'] = detections['detection_classes'].astype(np.int64)
71
72      label_id_offset = 1
73      image_np_with_detections = image_np.copy()
74
75      viz_utils.visualize_boxes_and_labels_on_image_array(
76                  image_np_with_detections,
77                  detections['detection_boxes'],
78                  detections['detection_classes']+label_id_offset,
79                  detections['detection_scores'],
80                  category_index,
81                  use_normalized_coordinates=True,
82                  max_boxes_to_draw=5,
83                  min_score_thresh=.5,
84                  agnostic_mode=False)
85
86      cv2.imshow('object detection',  cv2.resize(image_np_with_detections, (800, 600)))
87
88      if cv2.waitKey(1) & 0xFF == ord('q'):
89          cap.release()
90          break
```

A22. Creació aleatòria de conjunts de *train* i *test*

train_test.py

```
1   # import the necessary packages
2   from __future__ import print_function
3   from etaprogress.progress import ProgressBar
```

```
4   import sys
5   import time
6   import os
7   import cv2
8   import numpy as np
9   import os.path
10  from os import listdir
11  import argparse
12  import xml.etree.ElementTree as ET
13  import json
14  from datetime import datetime
15  import random
16  from shutil import copyfile
17
18  # construct the argument parser and parse the arguments
19  ap = argparse.ArgumentParser()
20  ap.add_argument("-i", "--input_dir", required=True,
21      help="path to input dir")
22  ap.add_argument("-o", "--output_dir", required=True,
23      help="path to output dir")
24  ap.add_argument("-ix", "--input_xml", required=True,
25      help="path to input xml dir")
26  ap.add_argument("-ox", "--output_xml", required=True,
27      help="path to outpùt xml dir")
28  ap.add_argument("-fr", "--ratio", required=False,
29      help="ratio over 1 train default 0.9")
30  ap.add_argument("-f", "--from", required=False,
31      help="from file")
32  ap.add_argument("-t", "--to", required=False,
33      help="to file")
34  args = vars(ap.parse_args())
35
36  PATH_TO_INPUT = args["input_dir"]
37  PATH_TO_OUTPUT = args["output_dir"]
38  PATH_TO_INPUT_XML = args["input_xml"]
39  PATH_TO_OUTPUT_XML = args["output_xml"]
```

```
40  RATIO = args["ratio"] if args["ratio"] else 0.9
41
42  files = listdir(PATH_TO_INPUT)
43  files.sort()
44
45  if args['to']:
46      files = files[:int(args['from'])]
47
48  if args['from']:
49      files = files[int(args['from']):]
50
51  random.shuffle(files)
52
53  train = int(RATIO * len(files))
54
55  bar = ProgressBar(len(files), max_width=100)
56
57  for i in range(0, train):
58      bar.numerator = i + 1
59      print(bar, end='\r')
60      file = files[i]
61      ann = file.split(".")[0] + ".xml"
62      copyfile(os.path.join(PATH_TO_INPUT, file), os.path.join(PATH_TO_OUTPUT,'train',file))
63      copyfile(os.path.join(PATH_TO_INPUT_XML, ann),
    os.path.join(PATH_TO_OUTPUT_XML,'train',ann))
64
65  for i in range(train, len(files)):
66      bar.numerator = i + 1
67      print(bar, end='\r')
68      file = files[i]
69      ann = file.split(".")[0] + ".xml"
70      copyfile(os.path.join(PATH_TO_INPUT, file), os.path.join(PATH_TO_OUTPUT,'test',file))
71      copyfile(os.path.join(PATH_TO_INPUT_XML, ann),
    os.path.join(PATH_TO_OUTPUT_XML,'test',ann))
72
73  print()
```

A23. Anotacions XML a TXT per entrenament Viola Jones

xml_to_txt.py

```
# import the necessary packages
from __future__ import print_function
from etaprogress.progress import ProgressBar
import os
import cv2
import numpy as np
import os.path
from os import listdir
import argparse
import xml.etree.ElementTree as ET

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-o", "--output_dir", required=True,
    help="path to output dir")
ap.add_argument("-x", "--xml_dir", required=True,
    help="path to xml dir")
ap.add_argument("-i", "--img_rel_path", required=True,
    help="relative path to img dir")
ap.add_argument("-f", "--from", required=False,
    help="from file")
ap.add_argument("-t", "--to", required=False,
    help="to file")
ap.add_argument("-tn", "--txt_name", required=False,
    help="filename txt")
args = vars(ap.parse_args())

PATH_TO_OUTPUT = args["output_dir"]
PATH_TO_XML = args["xml_dir"]
PATH_TO_IMG = args["img_rel_path"]
FILENAME = args["txt_name"] if args["txt_name"] else 'output'

files = listdir(PATH_TO_XML)
files.sort()
```

```
35
36  if args["to"]:
37      files = files[:int(args["to"])]
38  if args["from"]:
39      files = files[int(args["from"]):]
40
41  bar = ProgressBar(len(files), max_width=100)
42
43  pos = open(os.path.join(PATH_TO_OUTPUT, FILENAME + '.txt'), "w")
44  neg = open(os.path.join(PATH_TO_OUTPUT, FILENAME + '_neg' + '.txt'), "w")
45
46  for i, file in enumerate(files):
47      bar.numerator = i + 1
48      print(bar, end='\r')
49
50      tree = ET.parse(os.path.join(PATH_TO_XML, file))
51      root = tree.getroot()
52
53      ann = [os.path.join(PATH_TO_IMG, root.find('filename').text)]
54      bbs = []
55      count = 0
56      for cls, bb in zip(root.iter('name'), root.iter('bndbox')):
57          if not cls.text == "mask":
58              continue
59
60          count += 1
61          xmin = bb.find('xmin').text
62          ymin = bb.find('ymin').text
63          w = int(bb.find('xmax').text) - int(xmin)
64          h = int(bb.find('ymax').text) - int(ymin)
65
66          bbs.append(xmin)
67          bbs.append(ymin)
68          bbs.append(str(w))
69          bbs.append(str(h))
70
71      #print(ann, count, bbs)
```

```
72
73      if count < 1:
74          neg.write(ann[0] + '\n')
75          continue
76
77      ann = ann + [str(count)] + bbs
78      ann = ' '.join(ann)
79
80      pos.write(ann  + '\n')
81
82  pos.close()
83  neg.close()
84  print()
```

Annex B: Resultats

A continuació, es mostren les matrius de confusió per diferents configuracions de *datasets*. Aquestes matrius mostren les prediccions encertades i errades per cada tipus d'objecte en el conjunt de *test* de cada *dataset*. Una errada es considera tant si l'objecte no ha estat detectat com si s'ha classificat malament.

B1. Nas i Boca. 24.768 imatges

Configuració:

- Detecció: nas i boca.
- *Datasets:*
    - FFHQ: 4.500 + 500 imatges.
    - FFHQ Bright: 4.500 + 500 imatges.
    - FFHQ Background: 4.500 + 500 imatges.
    - FFHQ Background Bright: 4.500 + 500 imatges.
    - Webcam: 198 + 23 imatges.
    - Webcam Bright: 198 + 23 imatges.
    - FFMD: 3.893 + 433 imatges.

FFHQ		PREDICTION	
		Right	Wrong
REAL	mask	**397**	0
	no-mask	**101**	2
		99,60%	

FFHQ Bright		PREDICTION	
		Right	Wrong
REAL	mask	**389**	3
	no-mask	**100**	8
		97,80%	

FFHQ Background		PREDICTION	
		Right	Wrong
REAL	mask	**401**	8
	no-mask	**81**	10
		96,40%	

FFHQ Background Bright		PREDICTION	
		Right	Wrong
REAL	mask	**351**	43
	no-mask	**80**	26
		86,20%	

Webcam		PREDICTION	
		Right	Wrong
REAL	mask	**13**	7
	no-mask	**2**	1
		65,22%	

Webcam Bright		PREDICTION	
		Right	Wrong
REAL	mask	**9**	5
	no-mask	**5**	4
		60,87%	

FMDD		PREDICTION	
		Right	Wrong
REAL	mask	**24**	182
	no-mask	**35**	379
		9,52%	

TOTAL		PREDICTION	
		Right	Wrong
REAL	mask	**1584**	248
	no-mask	**404**	430
		74,57%	

Taula 2. Matrius de confusió pel cas B1.

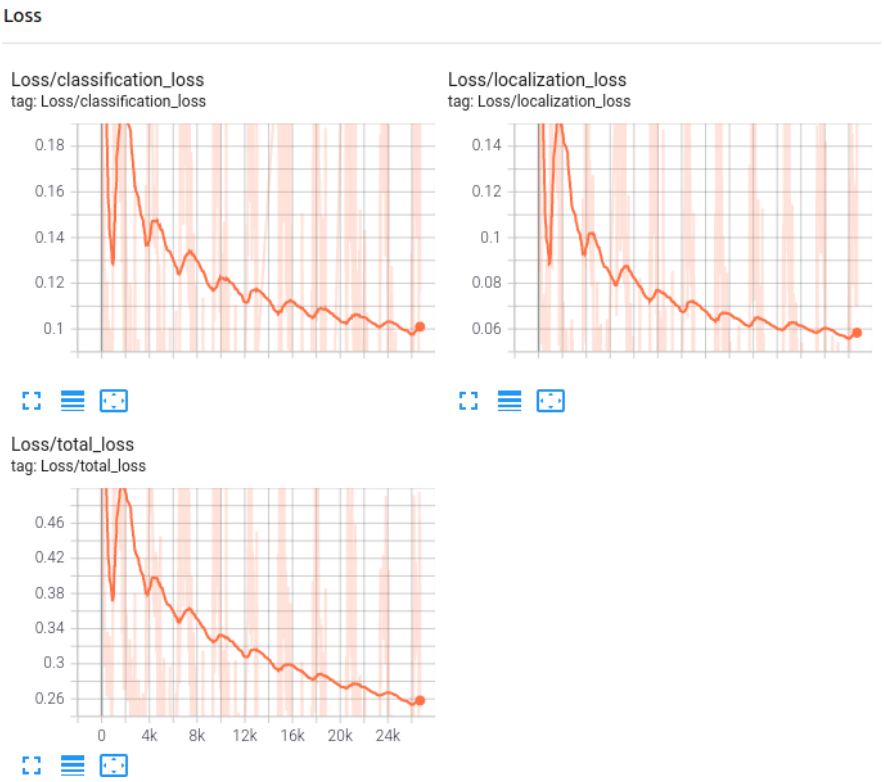

Figura 32. Corbes de pèrdua pel cas B1.

B2. Cara sencera. 24.768 imatges

Configuració:

- Detecció: nas i boca.
- *Datasets:*
  - FFHQ: 4.500 + 500 imatges.
  - FFHQ Bright: 4.500 + 500 imatges.
  - FFHQ Background: 4.500 + 500 imatges.
  - FFHQ Background Bright: 4.500 + 500 imatges.
  - Webcam: 198 + 23 imatges.
  - Webcam Bright: 198 + 23 imatges.
  - FFMD: 3.893 + 433 imatges.

FFHQ		PREDICTION	
		Right	Wrong
REAL	mask	**378**	5
	no-mask	**78**	39
		91,20%	

FFHQ Bright		PREDICTION	
		Right	Wrong
REAL	mask	**359**	30
	no-mask	**51**	60
		82,00%	

FFHQ Background		PREDICTION	
		Right	Wrong
REAL	mask	**369**	28
	no-mask	**78**	25
		89,40%	

FFHQ Background Bright		PREDICTION	
		Right	Wrong
REAL	mask	**311**	97
	no-mask	**35**	57
		69,20%	

Webcam		PREDICTION	
		Right	Wrong
REAL	mask	**8**	14
	no-mask	**0**	1
		34,78%	

Webcam Bright		PREDICTION	
		Right	Wrong
REAL	mask	**1**	13
	no-mask	**2**	7
		13,04%	

FMDD		PREDICTION	
		Right	Wrong
REAL	mask	**79**	389
	no-mask	**49**	278
		16,10%	

TOTAL	PREDICTION	
	Right	Wrong

REAL	mask	**1505**	576
	no-mask	**293**	467
		63,29%	

Taula 3. Matrius de confusió pel cas B2.

B3. Cara sencera. 344.581 imatges

Configuració:

- Detecció: cara sencera.
- *Datasets:*
    - FFHQ: 310.121 + 34.460 imatges.

FFHQ		PREDICTION	
		Right	Wrong
REAL	mask	**27547**	20
	no-mask	**6886**	7
		99,92%	

Taula 4. Matriu de confusió pel cas B3 i el seu conjunt de *test.*

Per altres conjunts amb els quals no s'ha entrenat aquest model els resultats són els següents:

FFHQ Bright		PREDICTION	
		Right	Wrong
REAL	mask	**334**	55
	no-mask	**98**	13
		86,40%	

FFHQ Background		PREDICTION	
		Right	Wrong
REAL	mask	**153**	244
	no-mask	**54**	49
		41,40%	

FFHQ Background Bright		PREDICTION	
		Right	Wrong
REAL	mask	**102**	306
	no-mask	**44**	48
		29,20%	

Webcam		PREDICTION	
		Right	Wrong
REAL	mask	**3**	19
	no-mask	**1**	0
		17,39%	

Webcam Bright		PREDICTION	
		Right	Wrong
REAL	mask	**2**	12
	no-mask	**5**	4
		30,43%	

FMDD		PREDICTION	
		Right	Wrong
REAL	mask	**41**	427
	no-mask	**36**	291
		9,69%	

TOTAL		PREDICTION	
		Right	Wrong
REAL	mask	**635**	1063
	no-mask	**238**	405
		37,29%	

Taula 5. Matrius de confusió pel cas B3 i altres conjunts amb els quals no s'ha entrenat.

B4. Cara sencera. 4.326 imatges

Configuració:

- Detecció: cara sencera.
- *Datasets:*
    - FFMD: 3.893 + 433 imatges.

FMDD		PREDICTION	
		Right	Wrong
REAL	mask	**313**	156
	no-mask	**196**	130
		64,03%	

Taula 6. Matriu de confusió pel cas B4 i el seu conjunt de *test.*

Per altres conjunts amb els quals no s'ha entrenat aquest model els resultats són els següents:

FFHQ		PREDICTION	
		Right	Wrong
REAL	mask	**344**	39
	no-mask	**116**	1
		92,00%	

FFHQ Bright		PREDICTION	
		Right	Wrong
REAL	mask	**307**	82
	no-mask	**81**	30
		77,60%	

FFHQ Background		PREDICTION	
		Right	Wrong
REAL	mask	**256**	141
	no-mask	**78**	25
		66,80%	

FFHQ Background Bright		PREDICTION	
		Right	Wrong
REAL	mask	**156**	252
	no-mask	**48**	44
		40,80%	

Webcam		PREDICTION	
		Right	Wrong
REAL	mask	**17**	5
	no-mask	**1**	0
		78,26%	

Webcam Bright		PREDICTION	
		Right	Wrong
REAL	mask	**7**	7
	no-mask	**5**	4
		52,17%	

TOTAL	PREDICTION	
	Right	Wrong

REAL	mask	**1400**	**682**
	no-mask	**525**	**234**
		67,76%	

Taula 7. Matrius de confusió pel cas B4 i altres conjunts amb els quals no s'ha entrenat.

Annex C: Comentaris sobre resultats

Els casos B1 i B2 s'han entrenat a partir dels mateixos conjunts d'imatges. El factor canviant és la zona d'anotació i detecció de l'objecte, pel cas B1 la xarxa detecta si la zona de boca i nas duu mascareta o no, mentre que pel cas B2 detecta tota la cara.

Comparant els resultats obtinguts es pot observar que la detecció de nas i boca és pràcticament més efectiva en gairebé tots els conjunts, suposant així una diferència més gran d'un 10% en el valor de rendiment total a favor de B1.

S'observa que B1 particularment té una significativa millor usabilitat per les imatges capturades amb webcam, factor important perquè en els objectius de la detecció preestablerts es volia tenir una bona usabilitat per aquest tipus d'imatges.

En la Figura 32 es pot observar l'avenç exponencial pel cas B1 en la reducció de les tres pèrdues valorades per l'eina Tensorboard. Això indica un bon ritme d'aprenentatge i garanteix certs resultats en les deteccions de prova. Les corbes per la resta de casos han estat similars.

D'altra banda, el cas B3, entrenat amb una gran quantitat d'imatges d'un mateix conjunt, presenta un rendiment gairebé perfecte pel seu conjunt de *test.* En veure aquests resultats es pot sospitar d'un èxit massa exagerat, essent fruit d'un entrenament *overfitted* o massa entrenat. Es pot comprovar aquest problema avaluant la xarxa per altres conjunts i veure el pobre rendiment, per exemple, en webcam.

Per últim, el cas B4, entrenat únicament amb imatges d'entorns quotidians com el carrer i el metro. Aquest *dataset* de caire difícil presenta poc més d'un 60% d'encert per a les seves pròpies mostres, percentatge bastant baix. No obstant, gràcies a la diversitat d'imatges es mostra bastant polivalent a l'hora de ser utilitzat en altres conjunts d'imatges. Els casos B1 i B4 són els que millor usabilitat per webcam han demostrat.

Annex D: Resultats visuals

D1. Deteccions zona boca i nas

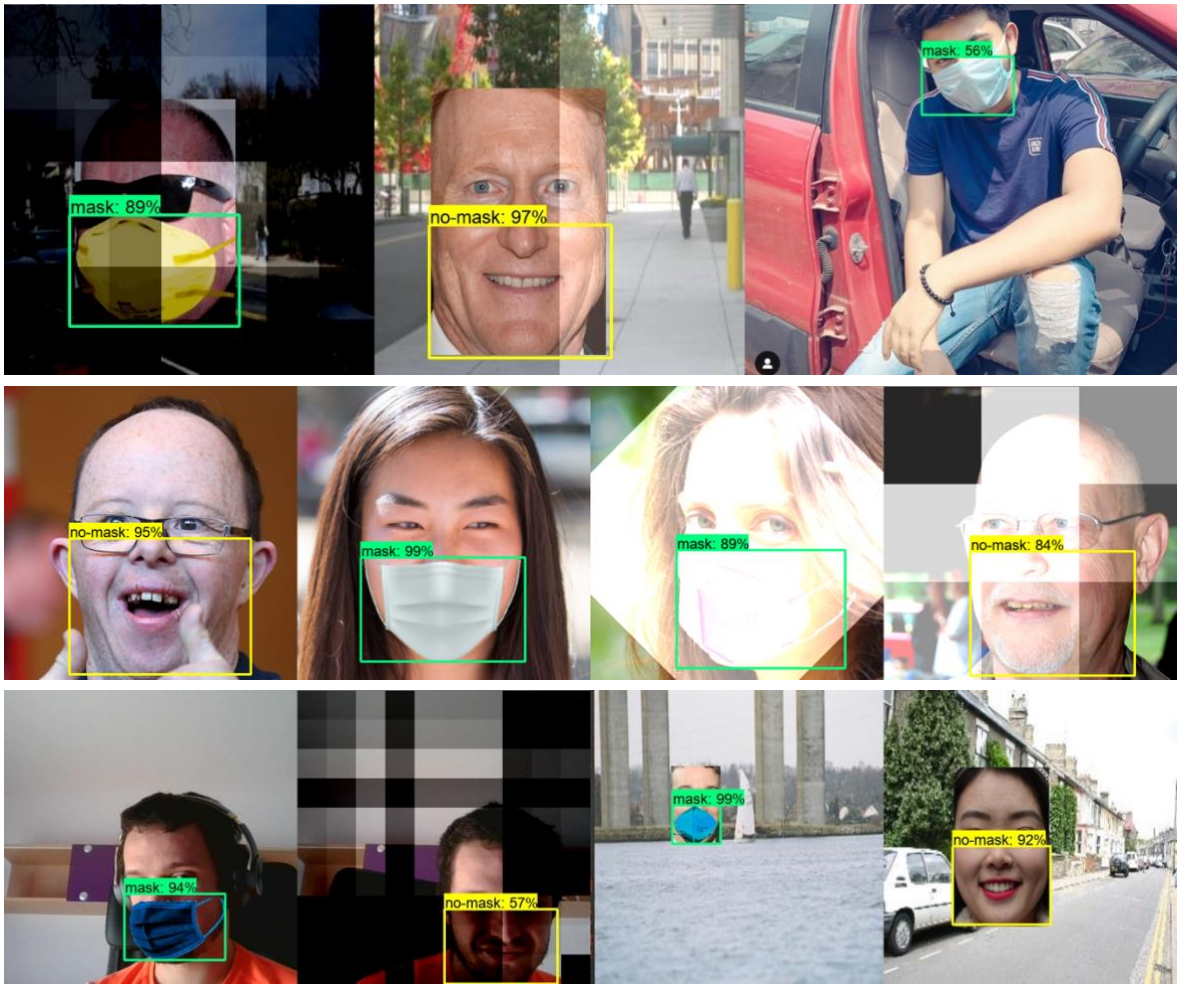

Figura 33. Exemples de detecció zona boca i nas.

D2. Deteccions cara sencera

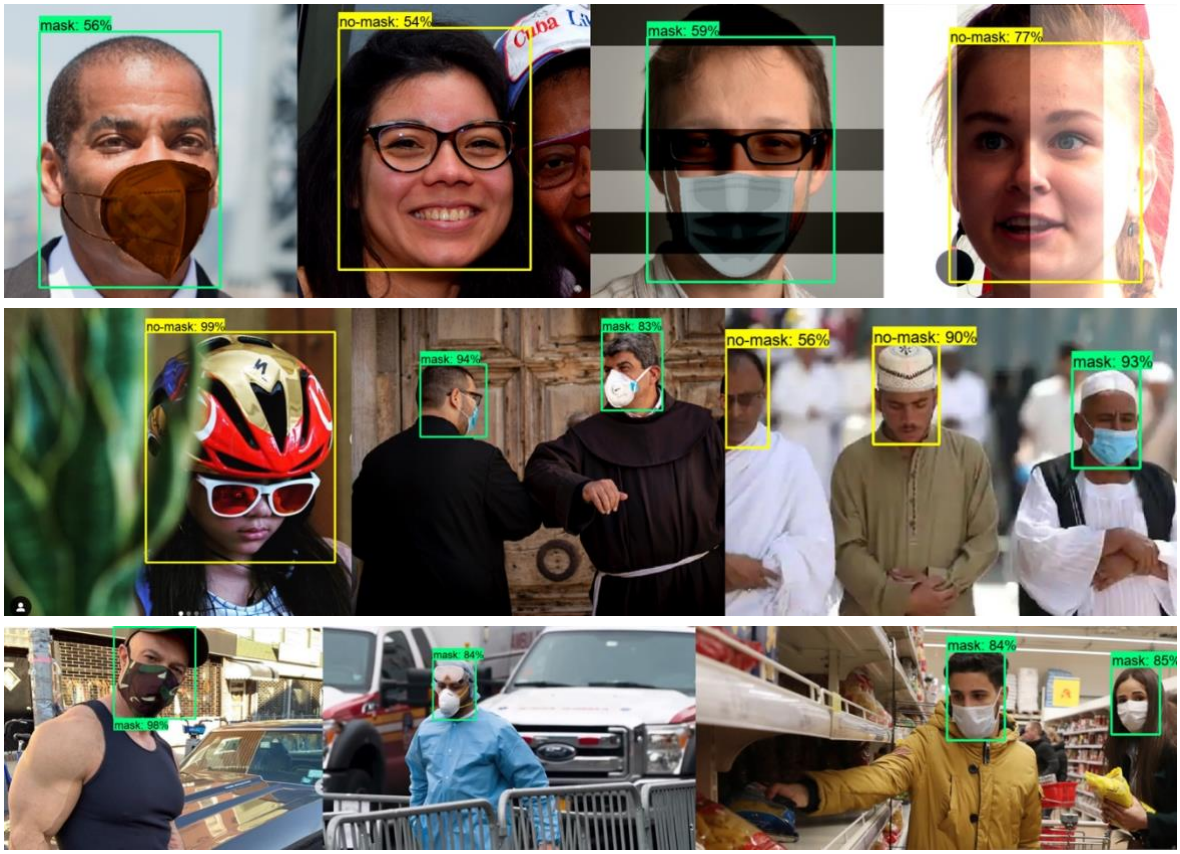

Figura 34. Exemples de detecció cara sencera.