Universitat Oberta de Catalunya

uoc.edu

Xarxes GAN i sistemes de seguretat basats en biometria facial

Joan Sierra Moreno
Màster Universitari en Ciberseguretat i Privadesa
M1.788 TFM – Anàlisi de dades

Enric Hernández Jiménez
Cristina Pérez Solà

Índex

1. Introducció

1.1 Context i justificació del Treball

La intel·ligència artificial s'ha convertit en una tecnologia que ens envolta en el nostre dia a dia. Els camps d'aplicació són molt diversos, des de la recerca predictiva d'alguns motors de recerca, passant per les recomanacions comercials, musicals o audiovisuals, fins als filtres per detectar correu brossa, per anomenar alguns exemples.

Les xarxes neuronals són el cor de la intel·ligència artificial, i durant els últims anys el seu desenvolupament ha crescut de forma molt notable. L'enfrontament de dues xarxes neuronals dona lloc a les xarxes GAN o xarxa adversativa generativa.

Una de les aplicacions més mediàtiques de les xarxes GAN ha sigut la creació d'imatges foto realistes de cares de persones totalment fictícies. A partir d'aquesta aplicació, i lligat als resultats tan espectaculars obtinguts[1], es qüestiona la possibilitat de fer servir una xarxa GAN amb finalitat no lícita.

Aquest TFM és una prova de concepte de com una xarxa GAN podria eludir un sistema d'autenticació basat en biometria facial, fent servir imatges totalment artificials de cares de persones, però amb les mateixes característiques biomètriques que les cares matriculades dins del sistema d'autenticació.

1.2 Objectius del Treball

Objectius d'investigació:

- Investigació sobre les xarxes neuronals.
- Investigació sobre conjunts de dades d'imatges d'àmbit públic.
- Investigació sobre els entorns de treball i de programació relatius a les xarxes neuronals.
- Investigació sobre sistema de biometria facial.
- Investigació sobre les xarxes GAN.
- Investigació sobre la creació de cares artificials amb una xarxa GAN.
- Investigació sobre la creació de cares artificials amb unes característiques biomètriques específiques.

Objectius d'implementació:

- Instal·lació de l'entorn de desenvolupament escollit.
- Implementació d'una xarxa neuronal que permeti identificar una cara d'una persona entre les cares matriculades.
- Implementació d'una xarxa neuronal que permeti la creació de cares de persones fictícies.
- Implementació d'una xarxa GAN que permeti crear cares de persones amb les mateixes característiques biomètriques que les cares de persones matriculades.

Objectius de les entregues:

- Entrega de la memòria.
- Entrega del vídeo.
- Defensa del TFM.

1.3 Enfocament i mètode seguit

En una primera etapa serà necessari adquirir els coneixements bàsics per entendre el funcionament de les xarxes neuronals i les seves diferents característiques i paràmetres d'optimització.

Una vegada adquirits aquests coneixements teòrics, la resta del projecte serà molt pràctic, generant diferents xarxes neuronals i avaluant els resultats obtinguts, per modificar posteriorment els paràmetres de les mateixes i aconseguir els resultats òptims. Aquesta operació de prova i error s'haurà de repetir vàries vegades, sent la capacitat de computació de l'equip utilitzat molt rellevant i crítica pel compliment dels terminis planificats.

Segons la dificultat per aconseguir els paràmetres òptims per cada xarxa neuronal, la planificació inicial podria veure's afectada.

Com ja s'ha indicat en els objectius d'implementació, abans de crear la xarxa GAN objectiu del TFM, s'hauran de crear dues xarxes neuronals amb resultats òptims, una pel reconeixement d'imatges de cares matriculades, i un altre per la creació de cares artificials.

Posteriorment, podrem fer la confrontació de les dues xarxes donant lloc a la xarxa GAN final.

1.4 Planificació del Treball

Recursos:

- Ordinador amb bona capacitat de computació:
  - CPU AMD Ryzen 7 – 3700 X – 8 cores (16 threads)
  - 32 Gb RAM DDR4 3200Mhz
  - GPU NVIDIA GTX1650 SUPER 6Gb – 1280 CUDA CORES

- Entorn de desenvolupament: *Google Colab PRO*
- Llibreries de programació específiques de *Deep Learning* i manipulació de dades.
- Base de dades de cares de persones, d'ús públic.

Tasques i planificació temporal:

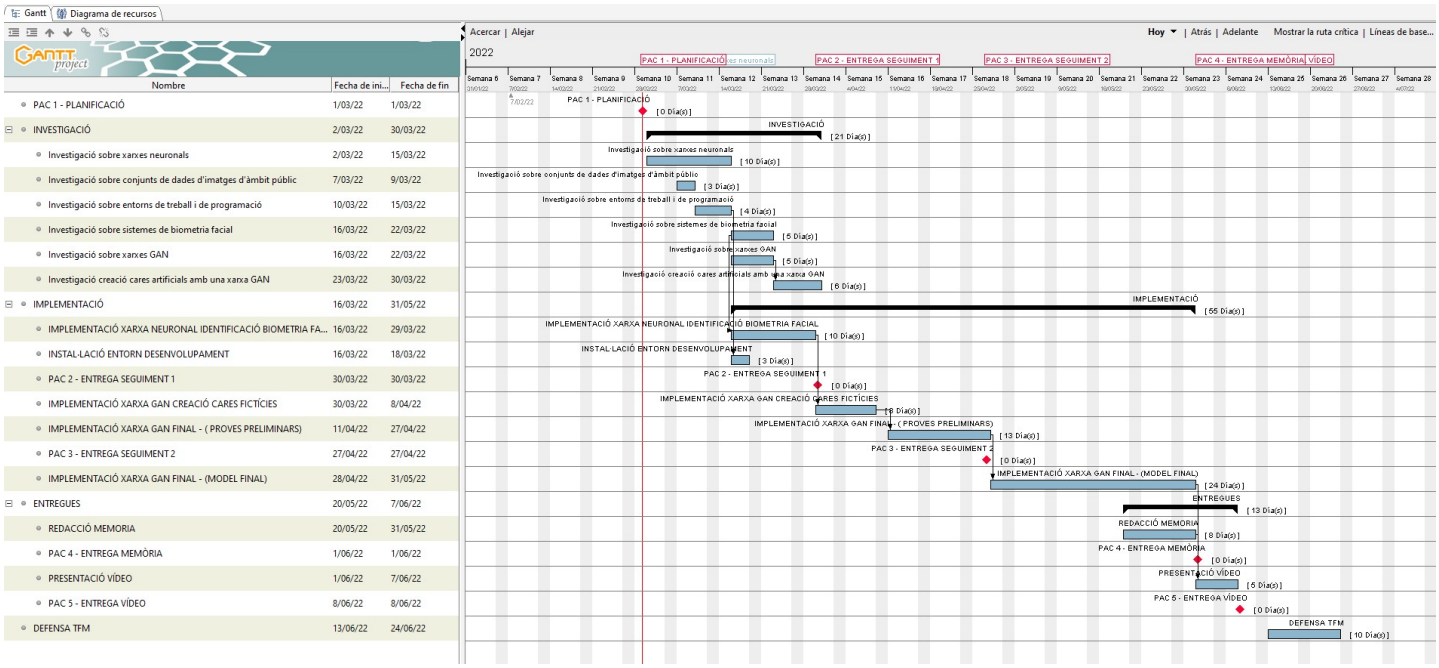


Fig. 1. Diagrama de Gantt del TFM

1.5 Breu sumari de productes obtinguts

Com a resultat haurem d'obtenir una xarxa neuronal (xarxa 1) capaç d'identificat imatges amb cares de persones en front d'imatges de cares matriculades.

Una segona xarxa neuronal del tipus GAN (XARXA 2) que sigui capaç de crear imatges de cares de persones i que alhora aquestes cares artificials puguin ser autenticades com matriculades per la xarxa neuronal inicial (XARXA 1).

2. Una incursió a la Intel·ligència Artificial i les seves variants

2.1 Intel·ligència Artificial

La intel·ligència artificial és un concepte força abstracte i no trivial que ens envolta en el nostre dia a dia, cada vegada més, com ja s'ha fet menció a l'apartat 1.1.

Altres exemples quotidians a part dels ja citats, podrien ser la traducció simultània oferta per *Google Translate*, el reconeixement de veu automàtic d'Alexa desenvolupat per *Amazon* o la conducció autònoma de vehicles que cada vegada ocupa més notícies en els medis.

Una aproximació a la definició d'IA podria ser la que trobem a la Viquipèdia:[2]

"La intel·ligència artificial (IA) és una part de la informàtica, dedicada al desenvolupament d'algorismes que permet a una màquina (habitualment un computador) prendre decisions intel·ligents o, si més no, comportar-se com si tingués una intel·ligència semblant a la humana".

Existeixen diferents treballs amb diferents definicions, però ens quedarem amb la idea general que la intel·ligència artificial és aconseguir que un sistema artificial tingui un comportament el més semblant possible al comportament humà, dins de l'àmbit de l'aprenentatge i presa de decisions.

2.2 *Machine Learning* (ML)

Aquest comportament l'aconseguim amb la creació d'algorismes que intenten 'simular' com pensem les persones. El conjunt de tècniques i algorismes que ho assoleixen és el que es coneix com a *Machine Learning* (ML) o tècniques d'aprenentatge automàtic.

Els algorismes de ML poden aprendre de forma autònoma els patrons existents en un conjunt de dades. Aquest aprenentatge és molt més eficient del que podríem fer la majoria d'humans, i com a resultat podem obtenir prediccions o classificacions d'una manera força eficient i acurades.

El ML es pot classificar en dos grans grups: aprenentatge supervisat i no supervisat, encara que existeixen altres possibles subclasses com l'aprenentatge per reforç o l'aprenentatge auto supervisat.

2.3 Deep Learning (DL)

Com podem veure a la figura 2, el ML és una part de la IA, i alhora el *Deep Learning* (DL) o aprenentatge profund és una part del ML.

Els algorismes que es fan servir al DL, a diferència dels del ML, estan basats majoritàriament en Xarxes Neuronals Artificials (ANN de l'anglès *Artificial Neural Networks*). Les ANN intenten simular el funcionament del cervell humà construint

una xarxa de neurones interconnectades d'una forma anàloga a com ho estan al nostre cervell. De fet, això només és una analogia perquè s'entengui el concepte, ja que el nivell de complexitat del cervell humà està lluny de ser entès actualment i menys encarà simulat.

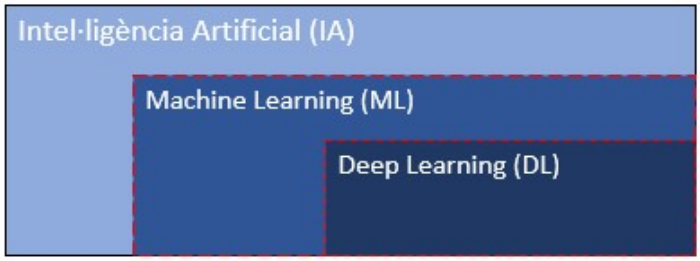


Figura 2. Esquema IA-ML-DL

2.3.1 Xarxes Neuronals Artificials

Una típica ANN està formada per diferents capes de neurones. Cada capa de neurones pot tenir el seu propi nombre de neurones. El nombre de capes, juntament amb el nombre de neurones de cada cap i com es connecten constitueixen, d'una manera simplificada, ja que hi ha altres paràmetres a considerar, el que es coneix com l'arquitectura de la ANN**.**

A la figura 2 podem veure una ANN formada per una capa d'entrada amb 2 neurones, una capa oculta amb 4 neurones i una capa de sortida amb una neurona. La ANN intenta resoldre un problema de classificació binària, separar els punts de color taronja dels blaus, i com veiem després de 5000 iteracions o èpoques **l'ANN** ha tingut un resultat força discret (*test loss*: 0.486), ha classificat correctament menys de la meitat dels punts.

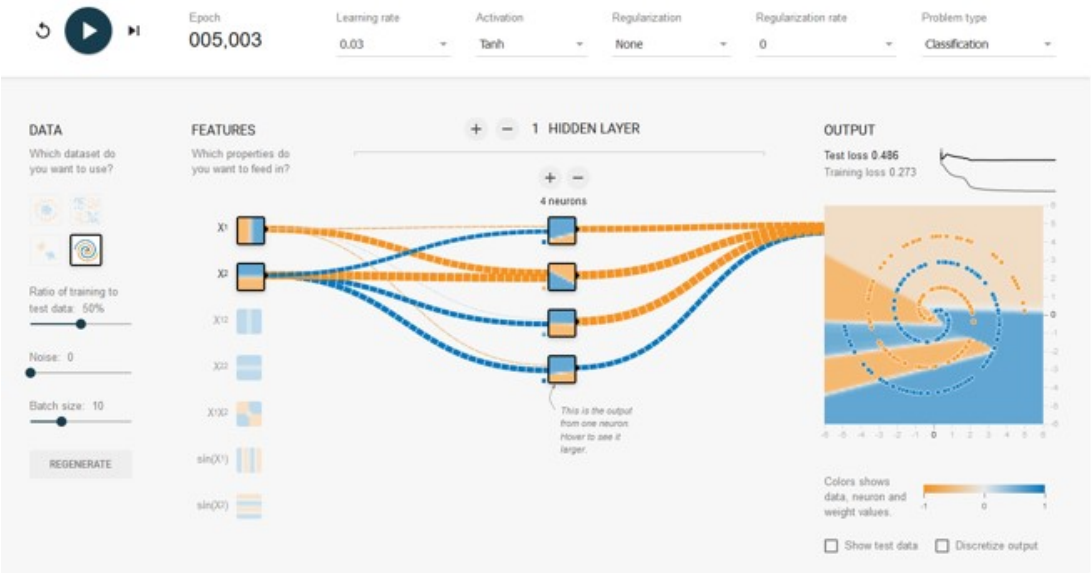


Figura 3. ANN 3 capes (1 entrada x2 neurones, 1 oculta x4 neurones, 1 sortida x1 neurona)
Https://playground.tensorflow.org

Si augmentem la complexitat o profunditat de l'ANN, com es mostra a la figura 4**,** amb 2 capes ocultes amb 6 i 4 neurones respectivament, i tractem de resoldre el mateix problema, veiem que el resultat aconseguit és força satisfactori (*Test Loss*: 0.007), aconseguint classificar correctament pràcticament tots esl punts.

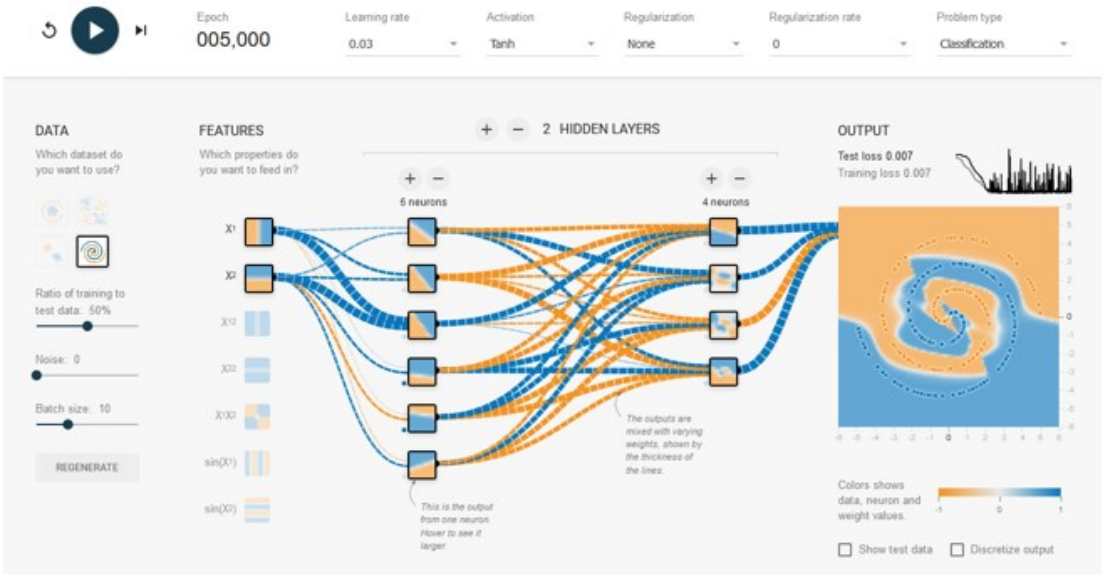


Figura 4. ANN 3 capes (1 entrada x2 neurones, 2 ocultes x6 i x4 neurones, 1 sortida x1 neurona)
Https://playground.tensorflow.org

A major profunditat major capacitat d'aprenentatge i, per tant, major capacitat per resoldre problemes no trivials. Però també necessitem més capacitat de computació i resoldre altres dificultats que sorgeixen en ANN profundes o molt profundes, com el sobre-entrenament. Actualment, trobem models que poden tenir des d'unes poques capes a models molt complexes com el RESNET1202 [3] format per 1202 capes.

Les ANN de les figures 3 i 4 han estat creades amb l'aplicació web interactiva ubicada a [4] creada per l'equip de desenvolupament de *Tensorflow* [5].

Cada una de les neurones d'una ANN pot rebre una o més entrades. A cada entrada d'informació si li aplica un pes (un valor que multiplica al valor d'entrada) i el resultat és computat amb una funció d'activació. Si el resultat d'aquesta funció satisfà el criteri definit per la mateixa funció, la neurona s'activa i passa el resultat a la següent capa de neurones.

Aquesta operació es repeteix a través de les diferents capes de neurones fins a arribar a la capa de sortida. Aquesta capa de sortida, habitualment té el mateix nombre de neurones que el problema que volem resoldre. Per exemple, si l'ANN ha de classificar imatges entre homes i dones, amb una única neurona a la capa final és suficient, ja que es tracta d'una classificació binària.

Les ANN fan servir majoritàriament mètodes d'aprenentatge supervisats, és a dir, hem d'aportar dades prèviament classificades a l'ANN perquè aquesta pugui aprendre. L'ANN fa servir aquestes dades per comparar el resultat obtingut amb

els paràmetres definits a la xarxa (per cada neurona el seu pes i funció d'activació), i en funció del resultat obtingut aplica una correcció sobre aquests paràmetres i torna a fer la mateixa operació de classificació i correcció de paràmetres de forma reiterativa fins que no es pot obtenir una millora en el grau d'exactitud de la classificació o predicció feta.

En aquest tipus d'ANN la informació flueix sempre en el mateix sentit, de l'entrada cap a la sortida, d'aquí que es coneguin amb el nom de *Feed-Forward Neuronal Networks (*FFNN*)*.

2.3.2 Xarxes Neuronals Convolucionals - CNN

Una variant de les ANN del tipus FFNN són aquelles xarxes que fan servir l'operació de convolució per fer extracció de patrons d'informació. Són les anomenades Xarxes Neuronals Convolucionals (CNN, *Convolutional Neuronal Networks*).

Aquestes xarxes s'han mostrat extraordinàriament potents en els sistemes de visió artificial i tractament d'imatges. Això és degut al fet que l'operació de convolució és una eina excel·lent per extreure i simplificar informació, convertint-se en un sistema d'extracció de característiques de les imatges amb un alt grau d'abstracció. Aquest grau d'abstracció va augmentant a poc a poc a mesura que ens endinsem en les diferents capes de la CNN. Habitualment, a cada capa amb una operació de convolució la segueix una o vàries capes que transformen el resultat de l'operació de convolució i l'adapten a l'entrada de la següent capa. Entre les operacions d'adaptació ens podem trobar capes que extreuen el valor màxim (*MaxPooling*), operacions de normalització (*BatchNormalization*), o de pèrdua de dades (*DropOut*) per citar algunes de les més utilitzades. A la figura 5 podem veure un exemple de CNN**,** desenvolupada per l'equip de NVIDIA[6], per poder llegir la velocitat màxima d'un senyal de trànsit. Podem veure que s'apliquen fins a quatre operacions de convolució en total.

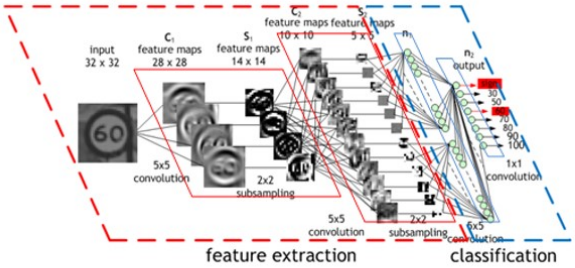


Figura 5. Esquema arquitectura xarxa CNN
https://developer.nvidia.com/discover/convolutional-neural-network

2.3.3 Models generatius

Els models generatius són un tipus d'algorisme de ML que ens permeten generar nous exemples similars a les dades originals a partir de l'extracció dels patrons existents a les dades disponibles. En aquest cas es tracta de models no supervisats. Aquest concepte de model generatiu dins del ML s'ha adaptat al DL donant lloc a la creació de diferents models com per exemple els auto codificadors variacionals(VAE [7], *Variational Auto Encoders*) o les xarxes generatives adversatives (GAN, *Generative Adversarial Networks*). Ens centrarem en aquestes últimes que són les que fan servir operacions de convolucions dins de la xarxa neuronal i a més s'han mostrat les més eficients en relació a la síntesi d'imatges.

2.3.4 Xarxes Generatives Adversatives (GAN)

La confrontació de dos CNN constitueix el que s'anomena una xarxa generativa adversativa (GAN**)**. En aquest tipus de xarxes, 2 CNN són confrontades mútuament per aconseguir l'aprenentatge del model generatiu. Això ho obtenim enfrontant dos models que tenen funcions molt específiques: un model generador i un model discriminador. El model va ser presentat per primera vegada l'any 2014 per *Ian Goodfellow* al *paper* "Generative Adversarial Networks"[8].

El model generador crea un exemple (en el nostre context parlarem únicament de creacions d'imatges) que alimenta al model discriminador. El model discriminador ha estat entrenat amb imatges reals. La classificació obtinguda pel discriminador ha de ser binària, imatge real o imatge falsa. Aquesta informació es propaga enrere tant al model discriminador com al generador. Els paràmetres de les dues xarxes són modificats en cada iteració i el procés es repeteix fins a aconseguir que el generador creï imatges que el discriminador no pot diferenciar de les reals.

A la figura 5b es pot veure d'una forma esquemàtica el seu funcionament.

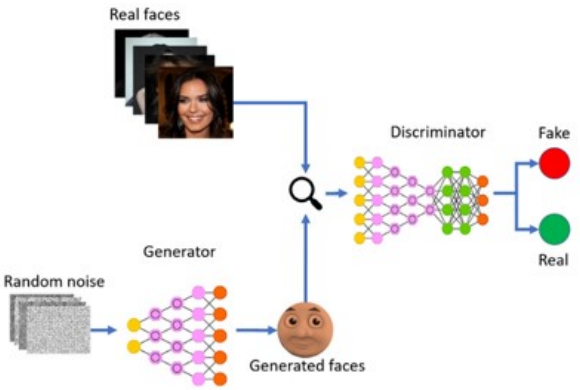


Figura 5b. Esquema d'una xarxa GAN per la creació de cares de persones inexistents (artificials)
https://medium.com/sigmoid/a-brief-introduction-to-gans-and-how-to-code-them-2620ee465c30

2.3.5 Xarxes GAN: l'Estat de l'Art

La increïble potència de les xarxes GAN ha fet que apareguin xarxes especialitzades en diferents àrees com la creació artificial de texts (*fakenews*), la creació d'imatges a partir d'un text, l'augment de la resolució d'imatges capturades amb poca qualitat, la creació d'art digital i la creació d'imatges foto realistes totalment artificials i inexistents a la realitat.

A la figura 6 podem veure una complexa taxonomia dels diferents àmbits d'investigació i desenvolupament que existeixen actualment.

Si ens centrem en la síntesi artificial d'imatges foto realistes, un dels models més conegut, gràcies als seus espectaculars resultats, és *StyleGan*, publicat l'any 2019 al *paper* "*A Style-Based Generator Architecture for Generative Adversarial Networks*" [9] per T.Karras, S.Laine i T.Aila de NVIDIA.

A diferència dels models previs, que es centraven en la millora de la xarxa discriminadora, *StyleGan* es centra en l'optimització de la xarxa generatriu.

Introdueix nous conceptes com la creació d'un espai latent de treball entremig, lligat a l'espai inicial que es rep a la capa d'entrada.

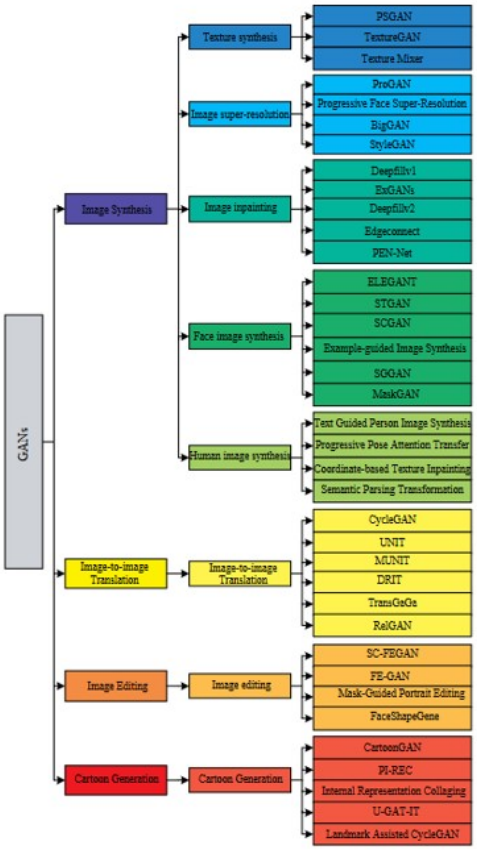


Figura 6. Taxonomia de les xarxes GAN.
https://www.researchgate.net/publication/340068457_A_State-of-the-Art_Review_on_Image_Synthesis_With_Generative_Adversarial_Networks/fulltext/5e756b06299bf1892cfbd582/A-State-of-the-Art-Review-on-Image-Synthesis-With-Generative-Adversarial-Networks.pdf?origin=publication_detail

Aquest nou enfocament juntament amb la introducció de soroll com una font de variació en cada punt del model generatriu, permet la generació d'imatges hiperrealistes amb control sobre diferents nivells de detall impensables anteriorment.

L'esquema del model *StyleGan* el podem veure a la figura 8, on es detalla el nou espai latent (*w*) i l'aplicació del soroll individual per cada estil obtingut (*B*).

Una mostra del que pot aconseguir StyleGan2 [10] la podem veure a la figura 7, una imatge fictícia creada amb StyleGan2 i que podem trobar a [11].

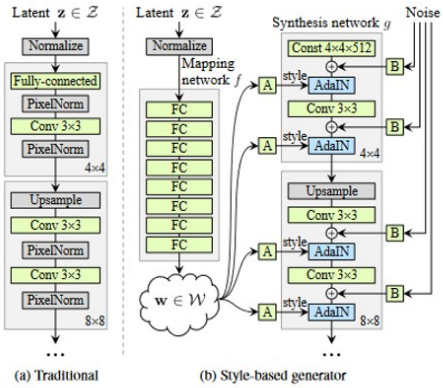


Figura 8. Esquema comparatiu StyleGan respecte a models tradicionals.
https://openaccess.thecvf.com/content_CVPR_2019/papers/Karras_A_Style-Based_Generator_Architecture_for_Generative_Adversarial_Networks_CVPR_2019_paper.pdf

L'octubre de 2021 es va fer públic StyleGan3 [12], la tercera versió que corregeix i millora substancialment el model inicial.

Figura 7. Imatge fotorealista creada amb StyleGan2. https://thispersondoesnotexist.com/

Avui en dia, amb molt poques línies de codi, podem generar les nostres pròpies imatges artificials gràcies a les diferents llibreries disponibles per entorns de treball com *Keras/Tensorflow* o *Pytorch* que implementen models complexos com *StyleGan*.

Per generar aquestes imatges necessitem un conjunt d'entrenament molt elevat, així com una gran capacitat de computació.

Aquests dos inconvenients els podem eludir fent servir tècniques de "*Transfer Learning" i "Fine Tunning*", que ens permeten fer servir models prèviament entrenats merament com extractors de característiques, per acabar d'entrenar la nostra xarxa amb el nostre conjunt de dades, molt més reduït, i obtenir uns resultats ràpids i espectaculars.

A la prova de concepte aprofitarem aquestes tècniques entre d'altres per aconseguir el resultat esperat: crear cares de persones fictícies que aconsegueixin eludir a un sistema biomètric d'identificació facial.

3. Sistema biomètric de verificació facial.

3.1 Funcionament

Els sistemes biomètrics de verificació funcionen comparant imatges de persones conegudes (registrades) amb les imatges de les persones que volen accedir al recinte o servei controlat amb el sistema. Les imatges de les persones autoritzades es registren i a partir d'aquestes imatges s'extreu la informació biomètrica fent servir diferents algorismes.

En el moment de fer una nova verificació, per exemple, una persona que vol accedir a un recinte controlat pel sistema, s'obtenen una o vàries imatges, s'extreuen els trets biomètrics característics i es comparen amb els trets biomètrics prèviament enregistrats de les persones autoritzades.

Quan les imatges es comparen, el que es fa és calcular la distància espacial entre els vectors que contenen els trets biomètrics. Si la diferència assolida està per sota d'un llindar específic, es pot garantir l'accés al recinte ja que considerem que hem fet una verificació positiva. En cas contrari, la diferència obtinguda és més gran que el llindar definit, la verificació es considera negativa i l'accés no s'obté.

El llindar de verificació és un paràmetre molt important en els sistemes biomètrics i en alguns casos difícil de definir d'una forma correcta. Si el llindar és molt baix, podem denegar l'accés a persones registrades. Si el llindar és molt alt, podem donar accés a persones no registrades i, per tant, no autoritzades.
En el primer dels casos comentats parlem de fals negatiu (llindar molt baix) i en el segon cas parlem de fals positiu (llindar molt alt).

3.2 Arquitectura

L'arquitectura del sistema biomètric escollit està basada en el model *SENET50* [13] prèviament entrenat amb el conjunt de dades *VGGFace2* [14], desenvolupat expressament pel reconeixement facial, i actualment un dels conjunts de dades que es podrien considerar l'estat de l'art dins d'aquest àmbit tan delimitat. Ha sigut creat pel *Visual Geometry Group (VGG) de la Universitat d'Oxford.*

Prèviament a l'ús del model farem servir un altre *CNN* considerada també l'estat de l'art: *MTCNN* [15]. Aquest model ens permet identificar i extreure les cares de persones d'una imatge. El resultat és un vector amb la ubicació de la cara dins de la imatge.

El procés que seguirem per construir el sistema biomètric de verificació facial el podem veure a la figura 9.

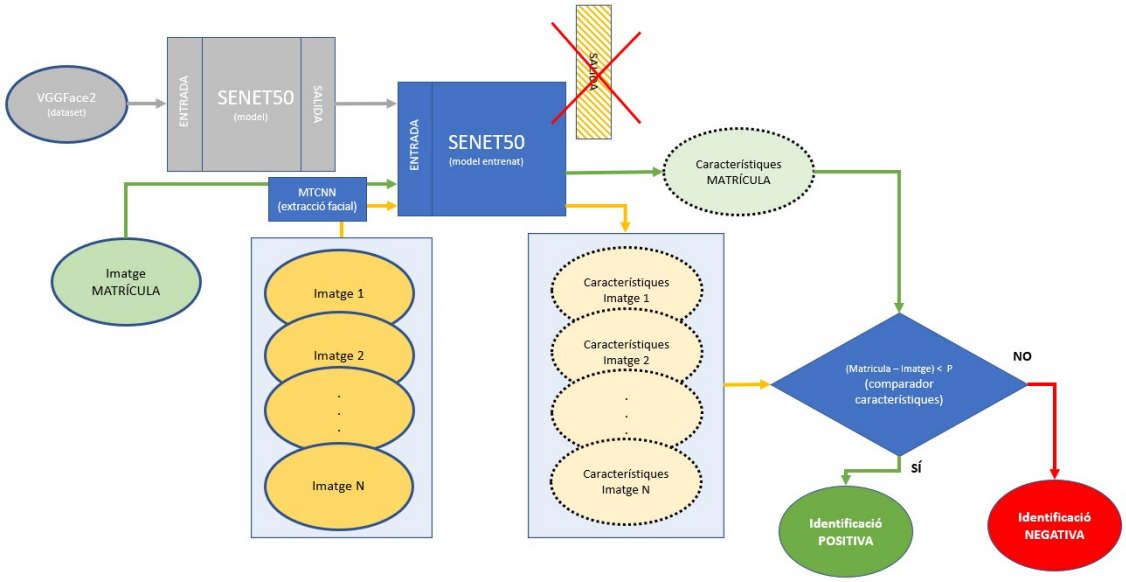


Figura 9. Arquitectura sistema biomètric de verificació facial proposat.

Carregarem el model sense la capa de sortida. D'aquesta manera la sortida del model seran les 'característiques' extretes de les noves imatges que passem com entrada al model. Aquestes imatges hauran estat prèviament tractades amb *MTCNN* per obtenir el vector de les mateixes que contenen només les cares detectades.
Per cada cara que alimentem al model, aquest ens retorna un conjunt de característiques pròpies de la cara detectada. En aquest punt podem registrar les cares de les persones 'autoritzades', és a dir, el procés de matriculació.

Per poder fer una verificació d'una altra persona, haurem de seguir el mateix procés, detectar la cara dins de la imatge, extreure el seu vector, retallar-lo i passar-lo com entrada al model *SENET50* que tenim ja entrenat amb el conjunt de dades *VGGFace2*.
Obtindrem les característiques d'aquesta nova persona i comparem amb les dades matriculades. La comparació que farem serà una diferència entre les característiques de la cara matriculada i la nova cara que hem de verificar. Si el resultat està per sota d'un llindar, que haurem de definir amb l'experiència, podrem dir que la identificació és positiva, altrament serà negativa.

3.3 Implementació

El codi es troba disponible en format *Jupyter Notebook* desenvolupat dins de la plataforma de *Google Colab* amb el nom *'PAC3_biometric_app.ipynb'*

S'intenta simular el que seria el funcionament d'un sistema biomètric de verificació facial com el descrit en l'apartat 3.1.

El primer que farem (cel·la #1) és clonar el repositori personal ubicat a *github* on tenim les imatges necessàries per poder demostrar la funcionalitat del codi. Això és necessari, ja que no podem desar de forma permanent arxius dins de la plataforma *Colab de Google*.

```
!git clone https://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@github.com/braludo/tfm.git

Cloning into 'tfm'...
remote: Enumerating objects: 236, done.
remote: Counting objects: 100% (76/76), done.
remote: Compressing objects: 100% (56/56), done.
remote: Total 236 (delta 16), reused 75 (delta 16), pack-reused 160
Receiving objects: 100% (236/236), 16.32 MiB | 3.43 MiB/s, done.
Resolving deltas: 100% (60/60), done.
```

Figura 10. Cel·la #1 codi Sistema Biomètric

Dins d'aquest repositori també està la versió corregida de l'arxiu models.py de *TensorFlow*. Aquesta versió resol un problema ja detectat amb la versió actual de *Tensorflow* i que en el moment d'escriure aquesta PAC estava pendent de '*commit'* en el repositori oficial.

A la cel·la #2 i #3 (figures 11 i 12) respectivament instal·lem totes les dependències necessàries i importem les llibreries que farem servir.

```
%%capture
## install interactive widgets
!pip install ipywidgets

# face verification with the SENET50 model pretrained with VGGFace2 dataset
!pip install mtcnn # face extractor
!pip install keras_vggface # VGG model
!pip install keras_applications # keras apps
```

Figura 11. Cel·la #2 codi Sistema Biomètric

```
## interact widgets to create dinamic app
%%capture
import ipywidgets as widgets
from ipywidgets import interact, interact_manual

import os, fnmatch
import tensorflow as tf

## replace original models.py file from tensorflow repo
## to solve open issue #81:  commit pending on github rcmalli/keras-vggface repo
## corrected models.py version is located on the github repo.

!cp tfm/resources/models.py /usr/local/lib/python3.7/dist-packages/keras_vggface/models.py

#importing libs

from matplotlib import pyplot
from PIL import Image
from numpy import asarray
from scipy.spatial.distance import cosine
from mtcnn.mtcnn import MTCNN
from keras_vggface.vggface import VGGFace
from keras_vggface.utils import preprocess_input
```

Figura 12. Cel·la #3 codi Sistema Biomètric

Per la primera part de l'aplicació farem servir una cel·la interactiva per escollir entre les imatges ubicades a dues carpetes diferents. En una carpeta tindrem les imatges registrades i a l'altra carpeta les imatges que contenen cares que volem verificar.
Escollirem del desplegable les imatges a comparar i obtindrem la diferència entre els trets característics d'ambdues imatges.

A la figura 13 podem veure el codi de la cel·la 4 corresponent a la funció '*get_embedding'* que ens retorna el vector que conté les característiques extretes.

```
## function to obtain embeddings from an extracted face
def get_embedding(face):
  pixels = face
  pixels = pixels.astype('float32')
  samples = tf.expand_dims(pixels, axis=0)
  samples = preprocess_input(samples,version=2)

  # create a vggface model
  # uncomment line to use different models
  # beeter results obtained with senet50 model
  #model = VGGFace(model='resnet50', include_top=False, input_shape=(224, 224, 3), pooling='avg') # original
  #model = VGGFace(model='vgg16', include_top=False, input_shape=(224, 224, 3), pooling='avg') # only for VGGF
  model = VGGFace(model='senet50',
                  include_top=False, input_shape=(224, 224, 3), pooling='avg') # better results for matches

  # perform prediction
  yhat = model.predict(samples)
  return yhat
```

Figura 13. Cel·la #4 codi Sistema Biomètric

A la figura 14 podem veure el codi contingut a la cel·la #5. Es defineix la funció interactiva *show_images()* que mostri en dues llistes desplegables les imatges contingudes en dos directoris, que contenen respectivament les imatges registrades (només una en aquest exemple per simplicitat) i les imatges que volem verificar.

Definim una funció interna *extreu()* que rep la ruta a una imatge i la mida de la imatge a retornar i, retorna una imatge de la cara extreta de la imatge rebuda.
Cridem dues vegades a la funció, una per cada una de les imatges escollides a la llista desplegable. Dibuixem les cares extretes i cridem a la funció *get_embedding()* una vegada per cada extracció feta. Comparem la diferència entre les característiques que ens torna la funció anterior computant la distància espacial entre els dos vectors amb la funció *cosine()* de la llibreria *scipy.*

```
## create two image selectors to compare cosine spatial distance
@interact
def show_images(image_1=os.listdir('tfm/biometric_app/registered/'),
                image_2=os.listdir('tfm/biometric_app/faces/')):
  filename_1 = 'tfm/biometric_app/registered/' + image_1
  filename_2 = 'tfm/biometric_app/faces/' + image_2
  def extreu(filename,required_size=(224,224)):
    # load image from file
    pixels = pyplot.imread(filename) # load in read-binary mode
    # create the detector, using default weights
    detector = MTCNN()
    # detect faces in the image
    results = detector.detect_faces(pixels)
    # extract the bounding box from the first face
    x1, y1, width, height = results[0]['box']
    x2, y2 = x1 + width, y1 + height
    # extract the face
    face = pixels[y1:y2, x1:x2]
    # resize pixels to the model size
    image = Image.fromarray(face)
    image = image.resize(required_size)
    face_array = asarray(image)
    return face_array
  face_1 = extreu(filename_1)
  face_2 = extreu(filename_2)

  f = pyplot.figure()
  f.add_subplot(1,2,1)
  pyplot.imshow(face_1)
  f.add_subplot(1,2,2)
  pyplot.imshow(face_2)
  pyplot.show(block=True)
  emb_1 = get_embedding(face_1)
  emb_2 = get_embedding(face_2)

  print("Diferència entre imatges: ", cosine(emb_1, emb_2))
```

Figura 14. Cel·la #5 codi Sistema Biomètric

A la carpeta que conté les imatges a verificar existeixen diferents imatges de la mateixa persona (joan) en diferents angles, diferents expressions, amb ulleres i sense.

Si comparem aquestes imatges contra la imatge registrada, en el pitjor dels casos la diferència calculada és de 0.3868 (imatge joan_01 de la figura 15).

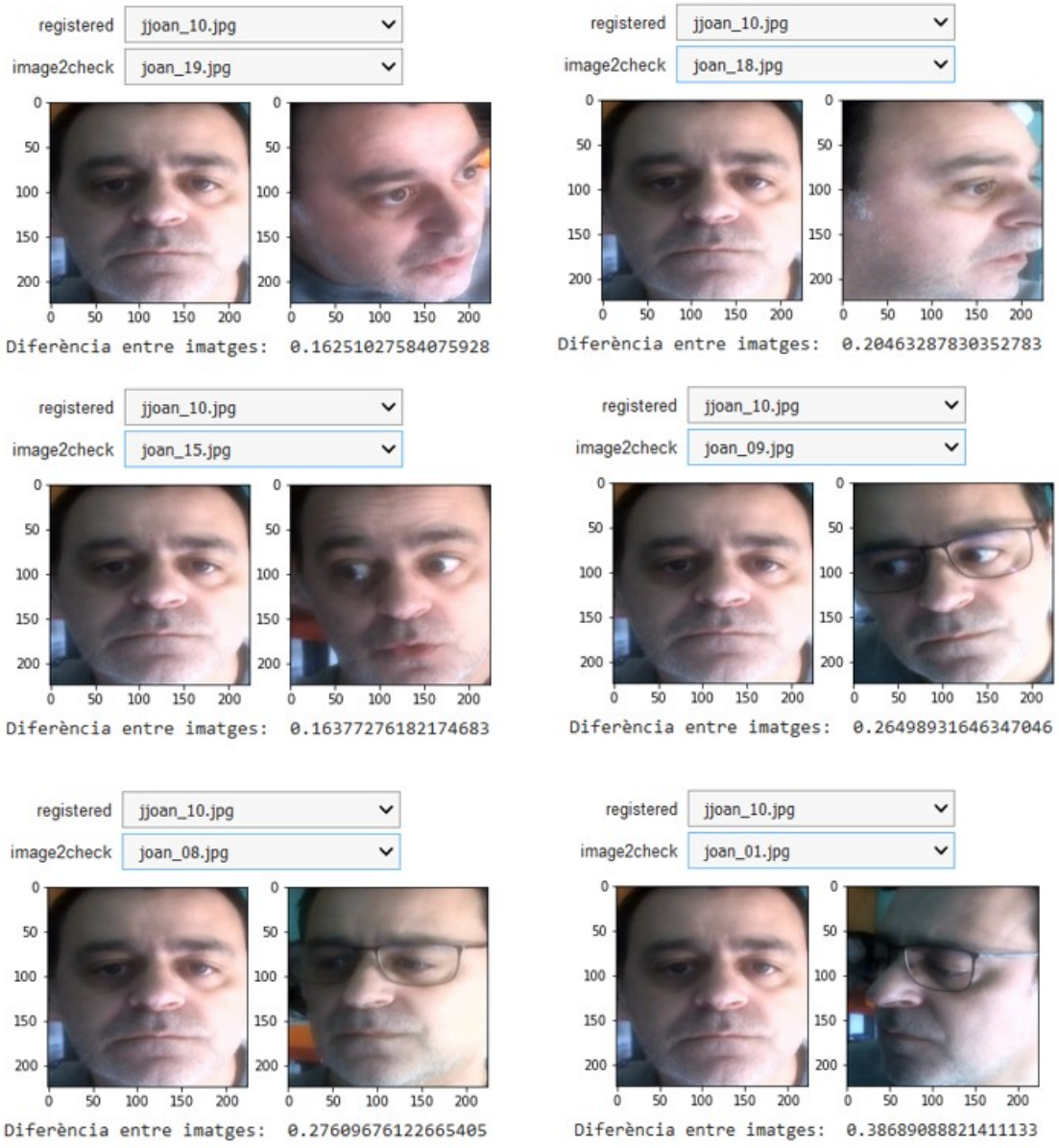


Figura 15. Diferències obtingudes entre imatges corresponents a la mateixa persona.

A la següent figura (figura 16) podem veure el resultat de comparar la imatge registrada contra la imatge d'altres persones. Podem veure que els valors obtinguts en tots els casos estan molt per sobre del 0.38 que acabem de veure. La imatge amb un valor més proper és la imatge *aina_01.jpg* amb un valor de 0.51.

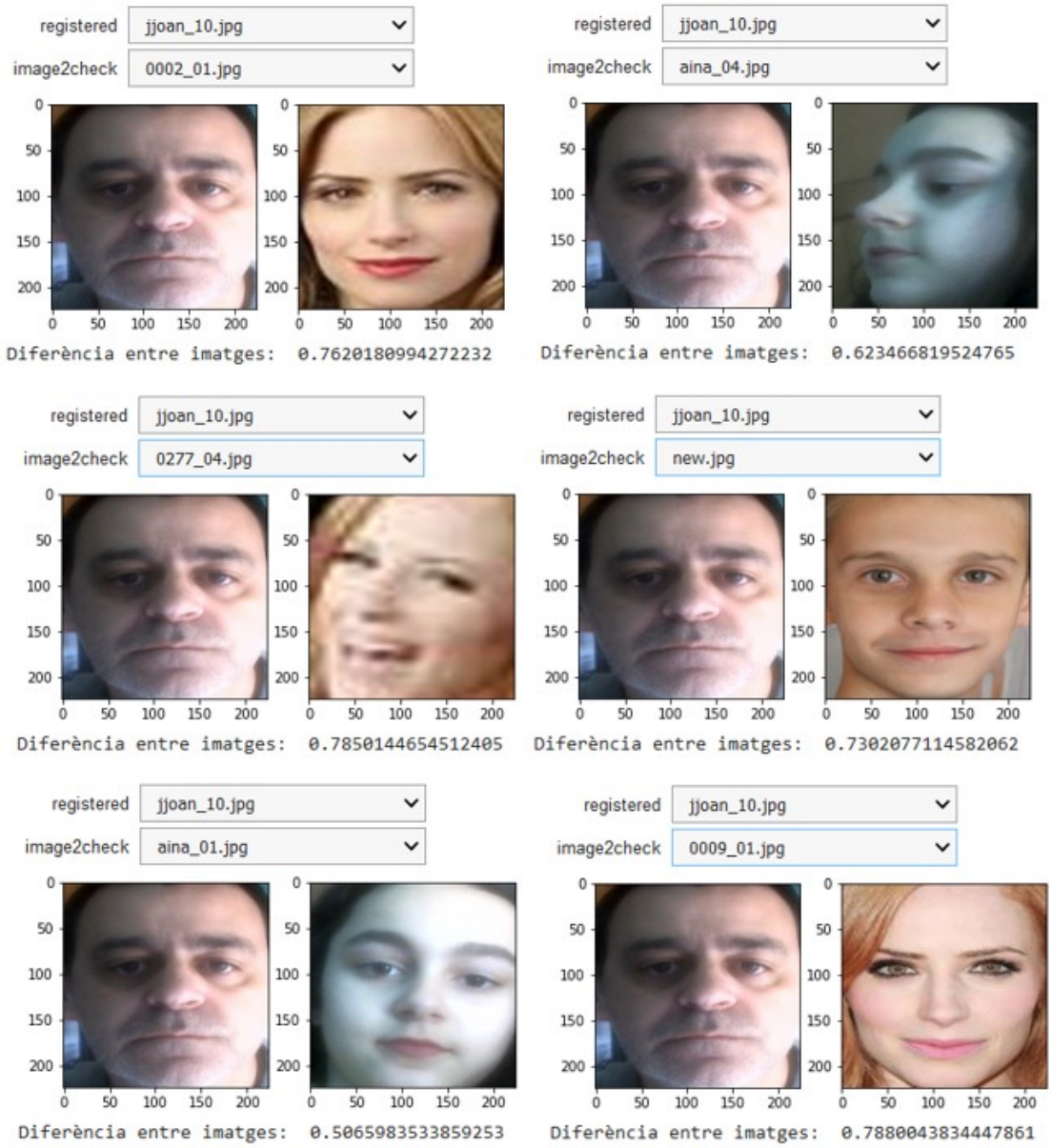


Figura 16. Diferències obtingudes entre imatges corresponents a la mateixa persona.

Veient aquests resultats podríem definir el llindar per una correcta verificació al voltant de **0.45.**

A la part 2 del codi s'introdueixen algunes petites modificacions i funcions noves per poder fer la comparació de forma automàtica entre la imatge de referència registrada i totes les imatges ubicades a la carpeta *tfm/faces/*

La figura 17 mostra el resultat per totes les imatges contingudes a la carpeta.

```
print ("__IMATGE__          __COSINE__")
for i in range(len(emb_to_check)):
  print(filenames[i], " ==> ", cosine(emb_registered, emb_to_check[i]) )
```

```
__IMATGE__          __COSINE__
0009_01.jpg  ==>  0.7880043685436249
joan_18.jpg  ==>  0.20463275909423828
aina_04.jpg  ==>  0.6234668493270874
joan_15.jpg  ==>  0.1637728214263916
joan_17.jpg  ==>  0.16849249601364136
joan_09.jpg  ==>  0.26498931646347046
0002_01.jpg  ==>  0.762018084526062
joan_19.jpg  ==>  0.16251027584075928
new.jpg  ==>  0.7302077114582062
aina_05.jpg  ==>  0.5587367117404938
joan_07.jpg  ==>  0.28598344326019287
0004_01.jpg  ==>  0.7672459334135056
aina_01.jpg  ==>  0.5065983831882477
joan_08.jpg  ==>  0.27609676122665405
joan_01.jpg  ==>  0.3868909478187561
joan_16.jpg  ==>  0.12124770879745483
aina_03.jpg  ==>  0.5828757286071777
joan_06.jpg  ==>  0.27728450298309326
joan_11.jpg  ==>  0.10174155235290527
joan_02.jpg  ==>  0.23841339349746704
joan_10.jpg  ==>  0.032028257846832275
0007_01.jpg  ==>  0.741124153137207
joan_05.jpg  ==>  0.2681800127029419
joan_04.jpg  ==>  0.2691923975944519
joan_03.jpg  ==>  0.1868608593940735
aina_02.jpg  ==>  0.5493521988391876
joan_12.jpg  ==>  0.11000645160675049
joan_14.jpg  ==>  0.13024485111236572
aina_06.jpg  ==>  0.5444863736629486
0277_04.jpg  ==>  0.7850144505500793
joan_13.jpg  ==>  0.04917776584625244
```

Figura 17. Diferències obtingudes entre imatges corresponents a la mateixa persona.

4. Generació facial orientada

Una vegada tenim definit i funcionant el sistema de biometria facial, ens cal generar cares artificials amb uns trets facials biomètrics semblants als de la imatge registrada. L'objectiu serà obtenir una imatge d'una cara de forma artificial a través d'una xarxa GAN, a partir de la que obtindrem el seu vector de característiques biomètriques tal que al calcular la diferència espacial fent servir l'aplicació de l'apartat 3, aquesta diferència sigui inferior al llindar (0.45) calculat a l'apartat 3.3 de la pàgina 17.

4.1 Progressive Growing GAN – ProGAN128

Farem servir una xarxa GAN del tipus 'creixement progressiu' (progressive growing). Les xarxes *ProGAN* van ser introduïdes per *T.Karras (Nvidia*) i van ser les precursores del model ja comentat *STYLEGAN* del mateix autor.
Bàsicament, el concepte que hi ha darrere és el de generar una imatge molt petita de partida i anar augmentant la mida de sortida a cada nova capa de la xarxa generativa dins del model GAN. El model proposat en el paper [16] dona com a resultat imatges foto realistes de 1024 x 1024 píxels de resolució fent servir com a conjunt de dades per l'aprenentatge el conegut *CELEBA*.

En el nostre codi farem servir la implementació reduïda disponible per *Tensorflow* que genera imatges de 128x128 píxels. El model preentrenat es troba disponible al *Hub* de *Tensorflow*, un repositori de models d'aprenentatge automàtic preentrenats.

Per escollir aquest model en concret entre tots els disponibles, s'ha tingut en consideració la baixa resolució de sortida de les imatges així com el conjunt de dades que s'han utilitzat durant l'entrenament (CELEBA). La baixa resolució ens permetrà poder fer un procés elaborat d'afinament de l'entrenament (*fine tunning*) sense necessitat d'una capacitat extraordinària de computació, i el conjunt de dades és molt apropiat per la prova de concepte motiu d'aquest treball.

4.2 Implementació

El codi el podem trobar en format *Jupyter Notebook* amb el nom '*PAC3_progan128.ypynb'* desenvolupat a la plataforma *Colab de Google* i consta de les cel·les següents.

A les cel·les #1 i #2 fem la còpia del repositori *Github* per carregar la imatge de referència i instal·lem les dependències necessàries (figura 18).

Figura 18. Codi cel·les #1 i #2 model generatiu progan-128.

A la cel·la #3 i #4 importem els paquets i definim les funcions auxiliars (figura 19).

CEL·LA # 3

Import needed packages

```
from absl import logging

import imageio
import PIL.Image
import matplotlib.pyplot as plt
import numpy as np

import tensorflow as tf
tf.random.set_seed(0)

import tensorflow_hub as hub
from tensorflow_docs.vis import embed
import time

try:
  from google.colab import files
except ImportError:
  pass

from IPython import display
from skimage import transform
```

CEL·LA # 4

Auxiliar functions

```
[ ]
# We could retrieve this value from module.get_input_shapes() if we didn't know
# beforehand which module we will be using.
latent_dim = 512


# Interpolates between two vectors that are non-zero and don't both lie on a
# line going through origin. First normalizes v2 to have the same norm as v1.
# Then interpolates between the two vectors on the hypersphere.
def interpolate_hypersphere(v1, v2, num_steps):
  v1_norm = tf.norm(v1)
  v2_norm = tf.norm(v2)
  v2_normalized = v2 * (v1_norm / v2_norm)

  vectors = []
  for step in range(num_steps):
    interpolated = v1 + (v2_normalized - v1) * step / (num_steps - 1)
    interpolated_norm = tf.norm(interpolated)
    interpolated_normalized = interpolated * (v1_norm / interpolated_norm)
    vectors.append(interpolated_normalized)
  return tf.stack(vectors)

# Simple way to display an image.
def display_image(image):
  image = tf.constant(image)
  image = tf.image.convert_image_dtype(image, tf.uint8)
  return PIL.Image.fromarray(image.numpy())

# Given a set of images, show an animation.
def animate(images):
  images = np.array(images)
  converted_images = np.clip(images * 255, 0, 255).astype(np.uint8)
  imageio.mimsave('./animation.gif', converted_images)
  return embed.embed_file('./animation.gif')

logging.set_verbosity(logging.ERROR)
```

Figura 19. Codi cel·les #3 i #4 model generatiu progan-128.

A les tres següents cel·les (#5, #6 i #7) carreguem el model progan128, la imatge de referència i la imatge *fake* de partida (figura 20).

CEL·LA # 5

Import progan128 model pretrained

```
[6]  progan = hub.load("https://tfhub.dev/google/progan-128/1").signatures['default']
```

CEL·LA # 6

Load reference image

```
image = tf.io.read_file('tfm_reference_image/jjoan_10.jpg')
image = tf.io.decode_image(image, channels=3, dtype=tf.dtypes.float32)
image = transform.resize(image, [128, 128])
target_image = image

display_image(target_image)
```

CEL·LA # 7

Load random genrated fake image

```
[ ]  tf.random.set_seed(1172)
     initial_vector = tf.random.normal([1, latent_dim])
     display_image(progan(initial_vector)['default'][0])
```

Figura 20. Codi cel·les #5 i #6 model generatiu progan-128.

A la cel·la #8 es defineix la funció que aplicarà l'ajustament fi (fine tunning) en el procéss d'aprenentatge utilitzant la tècnica de descens de gradient per trobar el vector latent generat dins de l'espai preentrenat més proper a la imatge de referència (figura 21).

CEL·LA # 8

Fine tunning function

```
def find_closest_latent_vector(initial_vector, num_optimization_steps,
                               steps_per_image, learning_rate, loss_function):
  images = []
  losses = []

  vector = tf.Variable(initial_vector)
  optimizer = tf.optimizers.Adam(learning_rate)
  loss_fn = loss_function

  for step in range(num_optimization_steps):
    if (step % 100)==0:
      print()

    print(step, '  ' , end='')
    with tf.GradientTape() as tape:
      image = progan(vector.read_value())['default'][0]
      if (step % steps_per_image) == 0:
        images.append(image.numpy())
      target_image_difference = loss_fn(image, target_image[:,:,:3])
      loss = target_image_difference
      losses.append(loss.numpy())
    grads = tape.gradient(loss, [vector])
    optimizer.apply_gradients(zip(grads, [vector]))
    #print(step, '  ' , spatial.distance.cosine(image, initial_vector), end='')

  return images, losses
```

Figura 21. Codi cel·la #8 model generatiu progan-128.Funció per fer 'fine tunning'

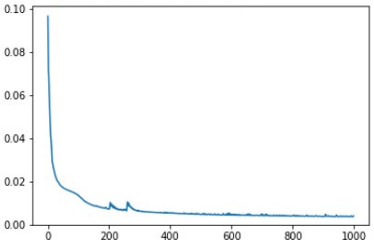
Aquesta funció és cridada des de la cel·la #9 a on configurem els diferents paràmetres: imatge inicial, nombre de passos pel cicle d'optimització, passos per imatge (per crear l'animació), ràtio d'aprenentatge i funció de pèrdua). La funció ens torna dos arrais amb les imatges generades durant l'aprenentatge i els valors de pèrdua calculats.
A les següents cel·les #10, #11 i #12 dibuixem la funció de pèrdua, les imatges inicial, convergent i referència i una animació des de la imatge de partida a la generada amb convergència (figura 22).

CEL·LA # 10

Plot loss function and lower value obtained

```
plt.plot(loss)
plt.ylim([0,max(plt.ylim())])
print('minim loss reached: ', min(loss))
```

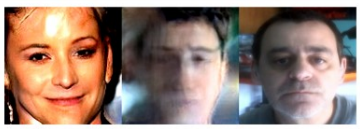
```
minim loss reached:  0.0036012246
```



CEL·LA # 11

Shows original fake face + converging fake face generated + reference face

```
[ ] display_image(np.concatenate([images[0], images[-1], target_image], axis=1))
```



CEL·LA # 12

Generate transition from original to converging faces

```
animate(np.stack(images))
```

Figura 22. Codi cel·la #8 model generatiu progan-128.Funció per fer 'fine tunning'

4.3 Minimitzant la pèrdua

Per poder aconseguir una imatge que pugui 'enganyar' al sistema de verificació biomètric, hem de generar una cara artificial que convergeixi fins a la imatge de referència. La imatge més propera serà la que s'obtingui amb una funció de pèrdua més petita, i per obtenir això haurem d'anar jugant amb els diferents paràmetres i avaluant la funció de pèrdua retornada.

Els paràmetres que haurem de configurar són els següents:

- Llavor (*seed)* de la imatge artificial aleatòria inicial.
- Nombre d'iteracions *(steps)*.
- Funció de pèrdua (*loss function).*
- Algosrisme optimitzador (*optimizer).*
- Ràtio d'aprenentatge (*learning rate)*.

Una de les possibles combinacions d'aquests 5 paràmetres ens donarà la millor configuració per resoldre el nostre problema, o si més no, aproximar-nos tot el possible. Per un altra banda, la limitació pel que fa a temps i capacitat de computació disponible, ha fet que a la bateria de proves realitzades s'hagin adoptat les següents consideracions o limitacions:

1- Fixar *seed* a un valor aleatòri, 1172 en el nostre cas.
2- Determinar la millor funció de pèrdua fixant la resta de paràmetres de forma arbitrària:
    - *steps:* 1000
    - *optimizer:* Adam
    - *learning rate:* 0.1

3- Amb la millor funció de pèrdua obtinguda provarem els diferents optimitzadors.
4- Amb la funció de pèrdua i optimitzador obtinguts augmentar les iteracions i disminuir el ratio d'aprenentatge.

Els resultats assolits per l'etapa 2 els podem veure a la taula de la figura 23. En verd hem identificat la funció de pèrdua que ens ha tornat un error més baix (les funcions identificades amb un asterisc retornen valors fora del rang 0 a 1 i s'han descartat a causa dels resultats aconseguits).

Com podem veure, la funció de pèrdua *MeanSquaredLogarithmicError()* ens ha retornat el valor més baix, per tant, fixarem aquesta funció per continuar amb l'etapa 3 i trobar el millor optimitzador.

A la taula de la figura 24 podem veure els resultats obtinguts pels diferents optimitzadors provats. De nou hem identificat en verd el millor algorisme d'optimització, la funció *Adamax*.

FUNCIÓ DE PÈRDUA	Valor minim de pèrdua obtingut
* MeanAbsolutePercentageError()	1728886,5
Hinge()	0,549
Poisson()	0,451
SquaredHinge()	0,384
BinaryCrossentropy()	0,314
CategoricalHinge()	0,178
MeanAbsoluteError()	0,054
BinaryFocalCrossentropy()	0,045
Huber	0,037
MeanSquaredError()	0,007
LogCosh()	0,004
MeanSquaredLogarithmicError()	0,003
* KLDivergence()	-0,481
* CosineSimilarity()	-0,996

Figura 23. Taula resultats amb diferent funció de pèrdua.

FUNCIÓ DE PÈRDUA: MeanSquaredLogarithmicError()	
OPTIMITZADOR	Valor minim de pèrdua obtingut
SGD	0,0263
RMSprop	0,0037
Adadelta	0,0264
Adagrad	0,0232
Adamax	0,0028
Nadam	0,0034
Ftrl	0,0277

Figura 24. Taula resultats amb diferent funció d'optimització.

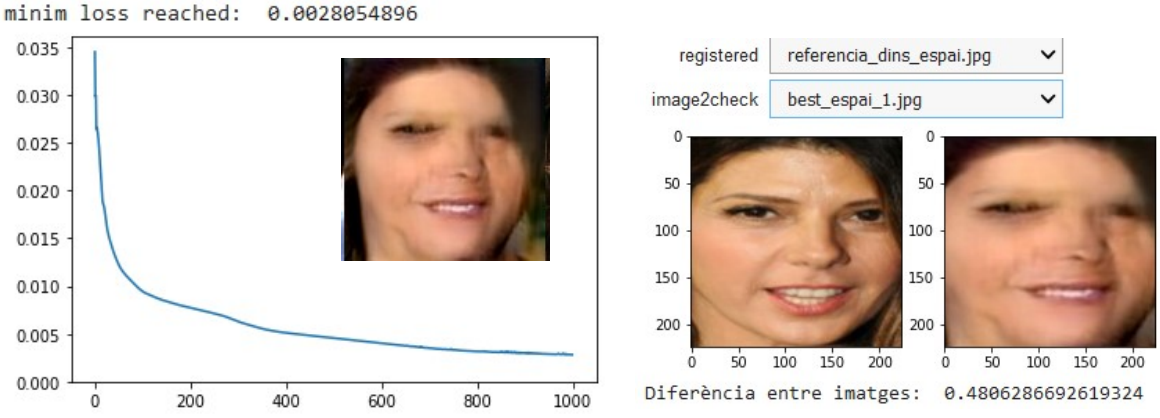
Una vegada arribats a aquest punt, només ens queda augmentar el nombre d'iteracions a uns valors assumibles computacionalment i reduir el ràtio d'aprenentatge. En el nostre cas únicament incrementarem les iteracions deixant el ràtio d'aprenentatge fixat a 0.1 pels mateixos motius explicats anteriorment.

Hem fet aquest exercici amb dues imatges de referència diferents, una d'una famosa ja inclosa dins del model preentrenat utilitzat (VGGFace2) i un altre fora d'aquest espai latent.

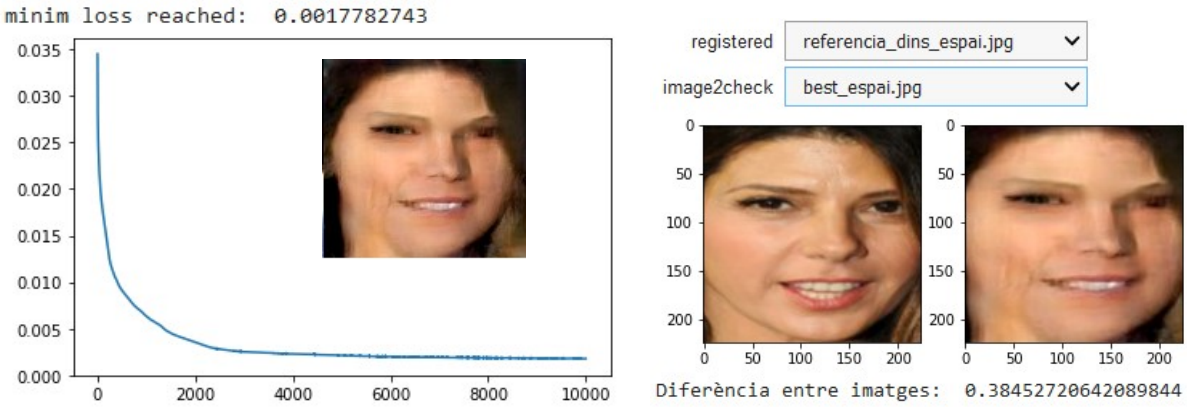
Veiem els resultats aplicant els paràmetres obtinguts anteriorment i augmentant progressivament el nombre d'iteracions de 1000 fins a 100000.

Per cada conjunt de proves detallem els paràmetres fets servir, la gràfica de la funció de pèrdua i el seu valor mínim obtingut, la imatge generada i el resultat de comparar la imatge de referència contra la imatge generada amb el nostre sistema de biometria facial.

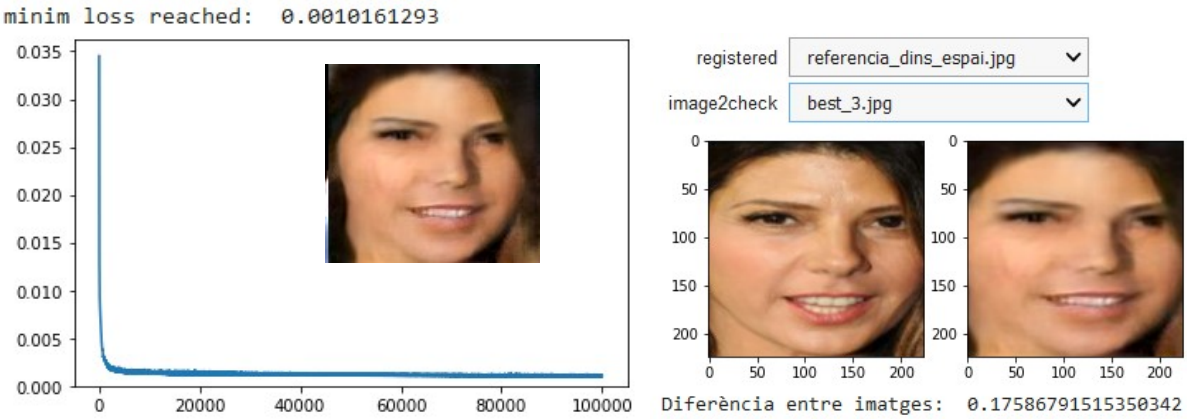
seed:	1152
funció de pèrdua:	MeanSquaredLogarithmicError()
optimitzador:	Adamax
ratio aprenentatge:	0.1
iteracions:	**1 000**



seed:	1152
funció de pèrdua:	MeanSquaredLogarithmicError()
optimitzador:	Adamax
ratio aprenentatge:	0.1
iteracions:	**10 000**



seed:	1152
funció de pèrdua:	MeanSquaredLogarithmicError()
optimitzador:	Adamax
ratio aprenentatge:	0.1
iteracions:	**100 000**



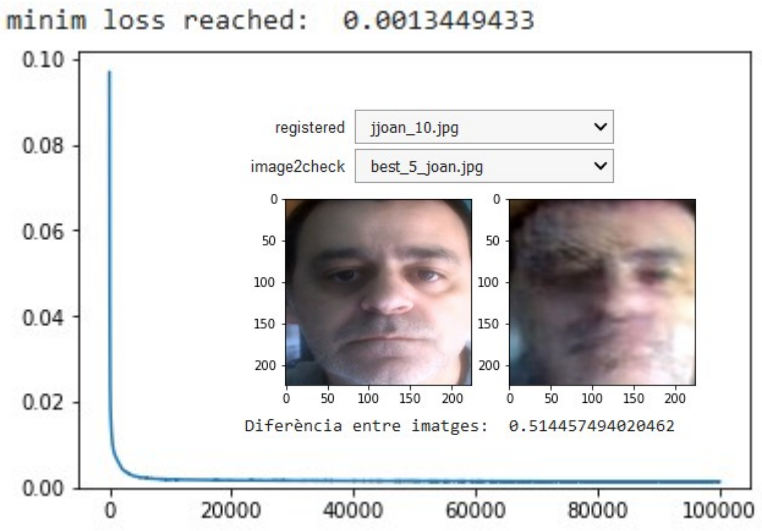
Podem veure que segons anem augmentant el nombre d'iteracions l'error que retorna la funció de pèrdua disminueix i conseqüentment la diferència respecte a la imatge de referència també, fins al punt que a partir de les 10000 iteracions la imatge obtinguda podria 'enganyar' al nostre sistema de verificació facial.

Recordem que el valor empíric que havíem definit per considerar que dues imatges pertanyen a la mateixa persona hauria de ser inferior a 0.45, condició que complim amb 10000 i 100000 iteracions, amb valors de 0.38 i 0.18 respectivament.

iteracions	error retornat	distància entre imatges (aplicació biomètrica)
1000	0,0028	0,48
10000	0,0018	0,38
100000	0,0010	0,18

Figura 25. Taula comparativa resultats final amb diferents iteracions.

Si bé el resultat obtingut amb una imatge de referència que pertany a l'espai latent generat ha sigut positiu, el cas contrari no ha donat resultats tan satisfactoris. Adjunt podem veure el resultat obtingut amb els mateixos paràmetres ja esmentats i 100000 iteracions:



L'error obtingut amb els mateixos paràmetres és molt similar en els dos casos; 0.0010 davant de 0.0013. Però la distància entre la imatge generada i la de referència és molt diferent; 0.18 davant de 0.51. En aquest cas no aconseguiríem 'enganyar' al sistema de biometria facial tot i que augmentant el nombre d'iteracions els resultats podrien millorar i arribar a ser inferiors al llindar de fals positiu del sistema biomètric (0.45). Com ja hem comentat anteriorment, no anirem més enllà a causa de l'alt cost computacional necessari.

El codi presentat en aquest treball ha estat reutilitzat o adaptat de les següents pàgines publicacions:

- https://machinelearningmastery.com/how-to-perform-face-recognition-with-vggface2-convolutional-neural-network-in-keras/
- https://machinelearningmastery.com/how-to-use-transfer-learning-when-developing-convolutional-neural-network-models/
- https://www.sitepoint.com/keras-face-detection-recognition/
- https://machinelearningmastery.com/how-to-train-a-progressive-growing-gan-in-keras-for-synthesizing-faces/
- https://www.tensorflow.org/hub/tutorials/tf_hub_generative_image_module

5. Conclusions

Les xarxes GAN han demostrat ser molt potents en diferents aspectes de la nostra vida com comentaven a la introducció. Hem fet una petita aproximació a la intel·ligència artificial, els seus conceptes i la tecnologia i algorismes que s'amaguen darrere.

Relacionant les IA amb la seguretat ens havíem proposat crear una prova de concepte per avaluar si un sistema *IA* basat en xarxes GAN seria capaç de crear imatges artificials prou realistes per poder 'enganyar' a un sistema de biometria facial.

Per fer-ho primer hem creat un sistema de biometria facial basat en l'estat de l'art de xarxes CNN per posteriorment crear les imatges fictícies amb una xarxa GAN preentrenada, amb un cost computacional relativament baix i assequible. Per fer-ho possible hem hagut de treballar amb imatges de molt poca resolució (128x128).

Hem demostrat el gran rendiment i capacitat de les xarxes GAN per generar imatges artificials de forma realista, i a més ho hem fet de forma orientada per generar imatges amb uns determinats trets facials, fins al punt de poder eludir un sistema de biometria facial.

La determinació dels paràmetres òptims de la xarxa GAN per crear les imatges no ha sigut trivial i ha fet consumir molt del temps de l'estudi.

Hem obtingut amb èxit imatges artificials que han donat com a resultat un fals positiu en el sistema biomètric partint d'una imatge de referència dins de l'espai latent generat per la xarxa GAN prèviament entrenada. Per contra, amb una imatge de referència de fora de l'espai latent preentrenat, els resultats no han sigut tan bons tot i que tampoc han sigut negatius, aconseguint una diferència respecte a la imatge de referència no molt lluny del llindar d'identificació.

Aquest últim fet ens demostra la gran importància de tenir un bon joc de dades per fer l'entrenament de la xarxa GAN.

Tot i que les imatges generades no són de molta qualitat degut a les limitacions computacionals ja esmentades, podem considerar la prova de concepte com a satisfactòria.

Per continuar aprofundint en la idea desenvolupada, seria necessari poder realitzar un estudi molt més exhaustiu generant imatges d´alta resolució i poder entrenar a la xarxa GAN amb joc de dades particulars en comptes de fer servir una xarxa preentrenada o inclús desenvolupar una xarxa GAN específica amb la finalitat de crear imatges amb determinats trets facials.

6. Glossari

TFM : Treball Final de Màster.
GAN: Generative Adversarial Network
IA: Intel·ligència Artificial
ML: Machine Learning
DL: Deep Learning
ANN: Artificial Neural Network
FFNN: Fast Forward Neural Network
CNN: Convolucional Neural Network
VAE: Variational Auto Encoders
GAN: Generative Adversarial Networks
VGG: Visual Geometric Group
MTCNN: Multi-task Cascaded Concolutional Networks
SENET50: Squeeze-and-Excitation Network (50 layers version)

7. Llista de figures

8. Referències

[1] *thispersondoesnotexist* [en línia] [consulta: 01 de març de 2022]. Disponible a : https://thispersondoesnotexist.com/

[2] https://ca.wikipedia.org/wiki/Intel%C2%B7lig%C3%A8ncia_artificial

[3] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770-778, doi: 10.1109/CVPR.2016.90.

[4] https://playground.tensorflow.org

[5] https://tensorflow.org

[6] *https://developer.nvidia.com/discover/convolutional-neural-network*

[7] Kingma, Diederik & Welling, Max. (2019). An Introduction to Variational Autoencoders. Foundations and Trends® in Machine Learning. 12. 307-392. 10.1561/2200000056.

[8] Generative Adversarial Nets. Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley,Sherjil Ozair, Aaron Courville, Yoshua Bengio

[9] A Style-Based Generator Aqrchitecture for Generative Adversarial Networks. T.Karras, S.Laine, T.Aila

[10] Analyzing and Improving the Image Quality of StyleGAN. T.Karras, S.Laine, T.Aila

[11] https://thispersondoesnotexist.com

[12] Alias-Free Generative Adversarial Networks. T.Karras, S.Laine, T.Aila

[13] Squeeze-and-Excitation Networks. J.Hu, L.Shen, G.Sun

[14] https://www.robots.ox.ac.uk/~vgg/publications/2018/Cao18/cao18.pdf

[15] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao, "Joint face detection and alignment using multitask cascaded convolutional networks," IEEE Signal Processing Letters, vol. 23, no. 10, pp. 1499–1503, Oct 2016.

[16] Progressive growing of gans for improved quality, stability, and variation. T.Karras, T.Aila, S.Laine, J.Lehtinen

9. Bibliografia

- **PYTHON DEEP LEARNING Introducción práctica con Keras y TensorFlow 2.** Jordi Torres – MARCOMBO.
- **Inteligencia Artificial, un enfoque moderno**. Stuart Rusell, Peter Norvig – PEARSON PRENTICE HALL
- **Deep Learning with TensorFlow**. Giancarlo Zaccone, MD. Rezaul Karim, Ahmed Menshawy – PACKT
- **Fundamentals of Deep Learning and Computer Vision**. Nikhil Singh, Paras Ahuja – BPB
- **GANs in Action: Deep learning with Generative Adversarial Networks**.Vladimir Bok,Jakub Langr – HANNING
- **Deep Learning with TensorFlow 2 and Keras - Second Edition: Regression, ConvNets, GANs, RNNs, NLP, and more with TensorFlow 2 and the Keras API, 2nd Edition.**Antonio Gulli, Amita Kapoor,Sujit Pal – PACKT
- **Convolutional Neural Networks in Python: Beginner's Guide to Convolutional Neu- ral Networks in Python.** Frank Millstein – SCRIBD.COM
- **https://machinelearningmastery.com/how-to-perform-face-recognition-with-vggface2-convolutional-neural-network-in-keras/**
- **https://machinelearningmastery.com/how-to-use-transfer-learning-when-developing-convolutional-neural-network-models/**
- **https://www.sitepoint.com/keras-face-detection-recognition/**
- **https://machinelearningmastery.com/how-to-train-a-progressive-growing-gan-in-keras-for-synthesizing-faces/**
- **https://www.tensorflow.org/hub/tutorials/tf_hub_generative_image_module**