Humanidades, Tecnología y Ciencia, del Instituto Politécnico Nacional
Certificado de Reserva de Derechos al Uso Exclusivo del Título: 04-2010-0326124-14000-203 ISSN: 2007-1957

REDES ADVERSARIAS PARA LA GENERACIÓN DE ROSTROS SINTÉTICOS USANDO PYTHON

Israel Rivera Zárate
Instituto Politécnico Nacional-CIDETEC
irivera@ipn.mx

Miguel Hernández Bolaños
Instituto Politécnico Nacional-CIDETEC
mbolanos@ipn.mx

Patricia Pérez Romero
Instituto Politécnico Nacional-CIDETEC
promerop@ipn.mx

Abstract

Neural networks have been used mainly for feature extraction and classification tasks with high dimensional distributions of data. Currently, networks known as generative adversarial networks (GAN) are emerging. This model can be characterized by training a pair of networks in competition with each other, the generator and the discriminator respectively. These networks are aimed at generating totally artificial images, which are highly similar to the images used during their training process without extensively annotated information. The possible GAN applications include style transfer, image superresolution, semantic image editing and image synthesis among others.

Palabras clave: Redes adversarias, modelo generador, modelo discriminador, entrenamiento, imágenes sintéticas, imágenes de rostros humanos, función de costo, matriz de confusión, muestras y espacio latente.

Las redes neuronales han sido utilizadas principalmente para clasificar datos o bien para obtener una representación compacta de esos datos como en el caso de las redes autoencoder. Actualmente surgen unas redes conocidas como adversarias o redes GAN (*generative adversarial networks*) por sus siglas en inglés. Estas redes están orientadas a la generación de imágenes totalmente artificiales, que son altamente semejantes, a las imágenes utilizadas durante su proceso de entrenamiento. Las redes GAN fueron propuestas por Ian Goodfellow en el año 2014. En una Red Adversaria se tienen dos modelos, compitiendo el uno contra el otro: un generador y un discriminador. La competencia

Humanidades, Tecnología y Ciencia, del Instituto Politécnico Nacional
Certificado de Reserva de Derechos al Uso Exclusivo del Título: 04-2010-0326124-14000-203 ISSN: 2007-1957

entre estos dos modelos se puede ver a través de una analogía. El generador es como un falsificador que intenta producir billetes falsos sin que estos sean detectados. Mientras que el discriminador es la policía que intenta detectar estos billetes falsos. Esta competición lleva a ambos equipos: falsificador y policía, o generador y discriminador, a mejorar sus métodos. En el caso de las redes adversarias se busca que al final sea el falsificador quien gane el juego, es decir, que logre finalmente engañar a la policía. Ver figura 1.

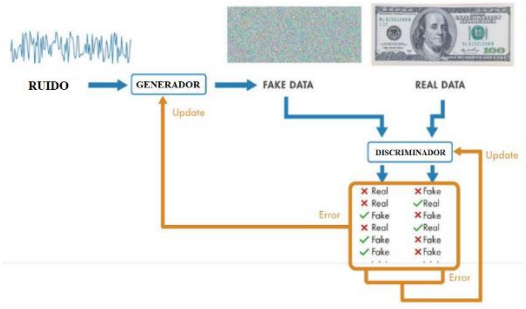


Figura 1. Redes adversarias generativas. Se observa el generador y el discriminador. Elaboración propia.

De acuerdo con Ian Goodfellow et al. (2014), en el contexto del Deep Learning, se entrenará un primer modelo: el discriminador D, que será capaz de reconocer rostros humanos. Este modelo puede ser un clasificador basado en una red convolucional en la que idealmente al ingresar una imagen con un rostro humano la salida será un 1, en tanto que de ingresar una imagen diferente la salida será 0. Esto se puede expresar como $D: D(x) \rightarrow (0,1)$. Representa una función que mapea los datos de una imagen hacia un valor de probabilidad. Adicionalmente, se llevará a cabo un segundo modelo: el generador G, cuyo objetivo es entrenarlo para recibir una entrada aleatoria: vector de ruido z ~ N(0,1) y a la salida producir algo muy parecido a la imagen de un rostro humano. Esto es $G: G(z) \rightarrow R^{|x|}$. Donde $z \in R^{|z|}$ es una muestra del espacio latente, $x \in R^{|x|}$ es una imagen, y $|.|$ corresponde al número de dimensiones. Estos dos modelos combinados reciben el nombre de redes adversarias. La idea es entrenar a estos dos modelos de forma simultánea buscando que al final sea el generador quien gane la competencia. El discriminador retroalimenta al generador, lo que le permite en cada pasada producir muestras más cercanas a las del entrenamiento.

Función de costo

Según lo indica Radford et al. (2015), se sabe que una red GAN ha terminado su proceso de entrenamiento cuando alcanza el equilibrio de Nash (problema de juegos de suma cero), donde la función de costo es una de entropía cruzada binaria que involucra las muestras reales y las muestras falsas. El discriminador $J^{(D)}$ busca minimizar los falsos positivos (FP) y los falsos negativos (FN) en la matriz de confusión entre la entrada de imágenes reales y falsas y su salida binaria. Así como el generador $J^{(G)}$ busca maximizar los falsos positivos (FP) ya que busca engañar al discriminador. Ver ecuación 1.

$$J^{(D)} = \underbrace{-\frac{1}{2}\mathbb{E}_{x\sim p_{data}}\log(D(x))}_{\text{Muestras reales}} \underbrace{-\frac{1}{2}\mathbb{E}_{z}\log\left(1-D(G(z))\right)}_{\text{Muestras falsas}} \quad \begin{matrix} D(x)\rightarrow 1 \\ D(G(z))\rightarrow 0 \end{matrix} \quad \ldots (1)$$

Al principio del entrenamiento las imágenes del generador no serán similares a las de un rostro, de modo que para el discriminador será fácil diferenciar las imágenes reales de las falsas (error pequeño). Sin embargo, a medida que avance el entrenamiento, el generador aprenderá poco a

Humanidades, Tecnología y Ciencia, del Instituto Politécnico Nacional
Certificado de Reserva de Derechos al Uso Exclusivo del Título: 04-2010-0326124-14000-203 ISSN: 2007-1957

poco a producir imágenes parecidas a un rostro humano. Por lo tanto, se logrará confundir al discriminador (error alto). Al final del entrenamiento, la salida del discriminador no será de 0 (imagen falsa), ni 1 (imagen real), sino 0.5, indicativo de que ha sido engañado por completo. El generador aprende entonces la distribución de los datos de entrada y por tanto aprende a replicar con precisión las imágenes de rostros humanos. Ver ecuación 2.

$$J^{(G)} = -\frac{1}{2}\mathbb{E}_z \log D(G(z))$$

$$D(G(z)) \to 1 \qquad \ldots (2)$$

I. Entrenamiento

Conforme lo indica Kingma et al. (2014), para cada iteración se hace:

A. Discriminador

1. Mini lote de ejemplos reales.

2. Mini lote de vectores de ruido que generan un mini lote de ejemplos falsos.

3. Se asigna la etiqueta 1 a los ejemplos de entrada reales y 0 a los falsos.

4. Se calcula el error. La función de costo penaliza los errores al clasificar las instancias reales como falsas y viceversa.

5. El error se propaga hacia atrás para actualizar los parámetros del discriminador.

B. Generador

1. Mini lote de vectores de ruido que genera un mini lote de ejemplos falsos.

2. Se asigna la etiqueta de 1 a los ejemplos falsos.

3. Se calcula el error.

4. El error se propaga hacia atrás para actualizar los parámetros del generador.

II. Metodología

La metodología adoptada en el presente trabajo consiste en 3 pasos esenciales:

Paso 1: Creación del modelo.

Paso 2: Generación de muestras del espacio latente.

Paso 3: Entrenamiento de la red GAN.

II.1. Creación del modelo

La implementación se desarrolló en Python (https://www.python.org, versión 3.6.1) con la ayuda de las librerías gratuitas PyTorch (https://pytorch.org, versión 2.0) y Tensorflow (https://www.tensorflow.org, versión 1.2.1), ampliamente aceptadas por la comunidad. El entorno de desarrollo empleado ha sido el Notebook de Jupyter (http://jupyter.org, versión 5.0.0).

a) Preparación del set de entrenamiento

Se utilizó el set de datos UTKFace de Kaggle (https://susanqq.github.io.UTKFace/) que contiene aproximadamente 2 mil imágenes con rostros humanos de edades ubicadas entre 0 y 116 años, cada una con un tamaño de 128x128 y que han sido

Humanidades, Tecnología y Ciencia, del Instituto Politécnico Nacional
Certificado de Reserva de Derechos al Uso Exclusivo del Título: 04-2010-0326124-14000-203 ISSN: 2007-1957

normalizadas en el rango de -1 a 1. Ver figura 2.

```
# Dataset from: https://susanqq.github.io/UTKFace/
import os
import numpy as np
import cv2
from PIL import Image
import random

n=20000 #Number of images to read from the directory. (For training)
SIZE = 128 #Resize images to this size
all_img_list = os.listdir('data/UTKFace/UTKFace/') #

dataset_list = random.sample(all_img_list, n) #Get n random images f

#Read images, resize and capture into a numpy array
dataset = []
for img in dataset_list:
    temp_img = cv2.imread("data/UTKFace/UTKFace/" + img)
    temp_img = cv2.cvtColor(temp_img, cv2.COLOR_BGR2RGB) #opencv rea
    temp_img = Image.fromarray(temp_img)
    temp_img = temp_img.resize((SIZE, SIZE)) #Resize
    dataset.append(np.array(temp_img))

dataset = np.array(dataset) #Convert the list to numpy array

#Rescale to [-1, 1] - remember that the generator uses tanh activati
dataset = dataset.astype('float32')
    # scale from [0,255] to [-1,1]
dataset = (dataset - 127.5) / 127.5
```

Figura 2. Fragmento conjunto de datos de entrenamiento. Elaboración propia.

b) Creación de la Red Adversaria

El Generador

Se tomará un vector con 100 números aleatorios y será entrenado para generar imágenes de 128x128 que progresivamente serán cada vez más parecidas a una imagen de un rostro. Ver figura 3.

La función inicia con un vector de 100 elementos que será conectado a una capa neuronal con 1024x4x4 elementos, que posteriormente será redimensionado a un volumen de 4x4x1024. Se utilizan los módulos Sequential y Dense. En las capas restantes se utiliza la convolución inversa (Conv2DTranspose) hasta progresivamente llegar al volumen deseado de 128x128x3 (es decir, las dimensiones de la imagen a generar). Ver figura 3.

```
def define_generator(latent_dim):
    model = Sequential()
    # Define number of nodes that can be gradually reshaped and upscaled to
    n_nodes = 128 * 8 * 8 #8192 nodes
    model.add(Dense(n_nodes, input_dim=latent_dim))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Reshape((8, 8, 128)))
    # upsample to 16x16
    model.add(Conv2DTranspose(128, (4,4), strides=(2,2), padding='same'))
    model.add(LeakyReLU(alpha=0.2))
    # upsample to 32x32
    model.add(Conv2DTranspose(128, (4,4), strides=(2,2), padding='same'))
    model.add(LeakyReLU(alpha=0.2))
    # upsample to 64x64
    model.add(Conv2DTranspose(128, (4,4), strides=(2,2), padding='same'))
    model.add(LeakyReLU(alpha=0.2))
    # upsample to 128x128
    model.add(Conv2DTranspose(128, (4,4), strides=(2,2), padding='same'))
    model.add(LeakyReLU(alpha=0.2))
    # output layer 128x128x3
    model.add(Conv2D(3, (8,8), activation='tanh', padding='same')) #tanh go
    return model
```

Figura 3. Fragmento de código de la red generadora. Elaboración propia.

Se observa en el código anterior que, en cada una de las capas, excepto por la de salida, ha usado la función de activación LeakyReLU, una variante de la función ReLU convencional que no elimina completamente los valores de entrada negativos. Del mismo modo, se ha usado *batch normalization*, que garantiza que a la salida de cada capa los valores tendrán un valor medio igual a cero y una desviación estándar igual a 1, lo que permitirá la convergencia del algoritmo de optimización durante el entrenamiento. Finalmente, en la capa de salida se emplea la función de activación tangente hiperbólica, para obtener imágenes generadas con pixeles en el rango de -1 a 1 (el mismo rango usado en las imágenes de entrenamiento reales). Este Generador será entrenado usando el algoritmo Adam que es una variante del Gradiente Descendente pero que requiere menos iteraciones para lograr la convergencia. El error para usar será la entropía cruzada, que es la misma métrica usada en la Regresión Logística, pues en este caso se tienen precisamente dos categorías (imagen falsa o imagen real).

El Discriminador

Será también una Red Convolucional prácticamente opuesta a la arquitectura del

Humanidades, Tecnología y Ciencia, del Instituto Politécnico Nacional
Certificado de Reserva de Derechos al Uso Exclusivo del Título: 04-2010-0326124-14000-203 ISSN: 2007-1957

Generador. Para ello se define en la entrada una imagen de 128x128x3 (el mismo tamaño de la imagen producida por el Generador) y en la salida mostrará un valor de probabilidad entre 0 y 1.

Se utilizarán las capas convolucionales Conv2d, con strides = 2 en lugar de capas MaxPooling que se encargarán de progresivamente reducir el tamaño de la entrada (ancho y alto) y de incrementar el número de características extraídas de la imagen, tal como lo hace una Red Convolucional convencional. Se propone utilizar BatchNorm2d en todas las capas del discriminador excepto en la entrada. Así mismo, se utilizarán las funciones de activación LeakyReLU, con una pendiente de 0.2 en las capas intermedias excepto en la salida que será sigmoidal para obtener valores entre 0 y 1 ya que el discriminador es un clasificador. Para entrenar este discriminador se utilizará el optimizador (ADAM) y la función de error de entropía cruzada. Ver figura 4.

```
# define the standalone discriminator model
# Input would be 128x128x3 images and the output would be a binary (using sigmoid)
#Remember that the discriminator is just a binary classifier for true/fake images.
def define_discriminator(in_shape=(128,128,3)):
    model = Sequential()
    # normal
    model.add(Conv2D(128, (3,3), padding='same', input_shape=in_shape))
    model.add(LeakyReLU(alpha=0.2))
    # downsample to 64x64
    model.add(Conv2D(128, (3,3), strides=(2,2), padding='same'))
    model.add(LeakyReLU(alpha=0.2))
    # downsample to 32x32
    model.add(Conv2D(128, (3,3), strides=(2,2), padding='same'))
    model.add(LeakyReLU(alpha=0.2))
    # downsample to 16x16
    model.add(Conv2D(128, (3,3), strides=(2,2), padding='same'))
    model.add(LeakyReLU(alpha=0.2))
    # downsample to 8x8
    model.add(Conv2D(128, (3,3), strides=(2,2), padding='same'))
    model.add(LeakyReLU(alpha=0.2))
    # classifier
    model.add(Flatten())
    model.add(Dropout(0.4))
    model.add(Dense(1, activation='sigmoid'))
    # compile model
    opt = Adam(lr=0.0002, beta_1=0.5)
    model.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
    return model
```

Figura 4. Fragmento de código de la red discriminadora. Elaboración propia.

Una vez creados el generador y el discriminador, resulta sencillo crear la Red Adversaria. Se utilizará inicialmente Sequential para crear el contenedor y luego se añadirá el generador y posteriormente el discriminador. Ver figura 5.

```
# define the combined generator and discriminator model, for updating the generator
def define_gan(g_model, d_model):
    # make weights in the discriminator not trainable
    d_model.trainable = False
    # connect them
    model = Sequential()
    # add generator
    model.add(g_model)
    # add the discriminator
    model.add(d_model)
    # compile model
    opt = Adam(lr=0.0002, beta_1=0.5)
    model.compile(loss='binary_crossentropy', optimizer=opt)
    return model

test_gan = define_gan(test_gen, test_discr)
print(test_gan.summary())
plot_model(test_gan, to_file='combined_model.png', show_shapes=True)
```

Figura 5. Fragmento de código de la GAN. Elaboración propia.

II.2. Generación de muestras del espacio latente

Para proporcionar muestras reales al modelo se utilizará el set de datos, sin embargo, para suministrar muestras falsas, se ha propuesto obtener imágenes a partir del espacio latente. El espacio latente es la representación compacta de la extracción de características obtenidas por parte de la red generadora a partir de las imágenes de entrada. El espacio latente es difícil de interpretar, por facilidad se emplean decenas de imágenes de una misma característica y se observa el lugar geométrico donde se mapean (hombre sonriendo, mujer neutral, hombre con barba, mujer con lentes, etc.), como se observa en la figura 6.

Humanidades, Tecnología y Ciencia, del Instituto Politécnico Nacional
Certificado de Reserva de Derechos al Uso Exclusivo del Título: 04-2010-0326124-14000-203 ISSN: 2007-1957

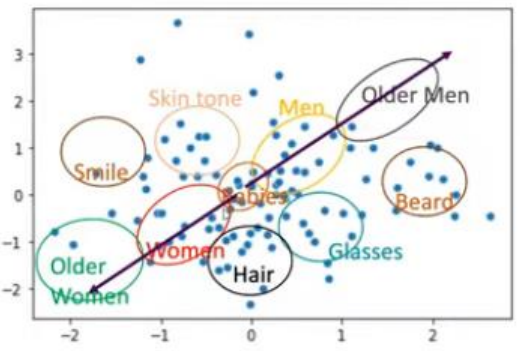


Figura 6. Espacio latente de la red generadora. Elaboración propia.

El espacio latente es sumamente curvo, sin embargo, por simplicidad se obtiene el promedio de los vectores latentes de cada categoría que representará una característica principal de la imagen. Los rostros pueden ser generados utilizando dos variables latentes, interpolando con base en la expresión del interpolador lineal, como se puede observar en la figura 7. Ver el fragmento de código en la figura 8.

$$z_{new} = z_A(1-\alpha) + z_B\alpha$$

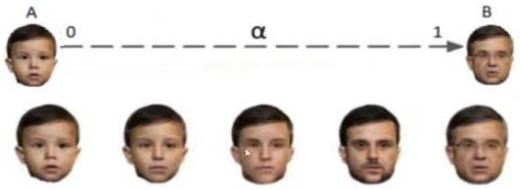


Figura 7. Interpolación lineal en el espacio latente. Elaboración propia.

```
#Interpolating between faces - Linear interpolation
#################################################################

from numpy import linspace

# Function to generate random latent points
#Same as defined above, re-defining for convenience.
def generate_latent_points(latent_dim, n_samples, n_classes=10):
    # generate points in the latent space
    x_input = randn(latent_dim * n_samples)
    z_input = x_input.reshape(n_samples, latent_dim) #Reshape to be
    return z_input

# Interpolation between two points in latent space
def interpolate_points(p1, p2, n_steps=10):
    # interpolate between points (e.g., between 0 and 1 if you divde
    ratios = linspace(0, 1, num=n_steps)
    # linear interpolation of vectors based on the above interpolati
    vectors = list()
    for ratio in ratios:
        v = (1.0 - ratio) * p1 + ratio * p2
        vectors.append(v)
    return asarray(vectors)

# create a plot of generated images
def plot_generated(examples, n):
```

Figura 8. Fragmento de código interpolación lineal. Elaboración propia.

Adicionalmente, se pueden generar imágenes con base en aritmética del espacio latente, donde es posible operar los vectores latentes, por ejemplo: “hombre sonriente”, “hombre con rostro neutral” y el de un “bebe con rostro neutral” y entonces generar un “bebe con rostro sonriente” mediante la operación suma y resta de vectores latentes respectivamente. Ver figura 9.

Figura 9. Aritmética en el espacio latente. Elaboración propia.

II.3. Entrenamiento de la Red Adversaria

Para el entrenamiento se utilizarán lotes de 128 imágenes y un total de 5000 iteraciones. En el caso de las Redes Adversarias se deben llevar a cabo los pasos siguientes:

1. Entrenar el Discriminador.

Humanidades, Tecnología y Ciencia, del Instituto Politécnico Nacional
Certificado de Reserva de Derechos al Uso Exclusivo del Título: 04-2010-0326124-14000-203 ISSN: 2007-1957

2. “Congelar” los coeficientes del Discriminador.

3. Entrenar únicamente el Generador.

4. “Descongelar” los coeficientes del Discriminador.

5. Repetir los pasos 1 a 4 por el número de iteraciones que se vayan a usar el entrenamiento.

Inicialmente se ha “congelado” el discriminador (usando la línea de código discriminador. trainable = False), para activar y desactivar su entrenamiento en cada iteración. La intención es generar una competencia entre los dos modelos: el generador y el discriminador, con el objetivo de que el generador logre incrementar el valor del error del discriminador, es decir, que logre confundirlo. Como en cada iteración se deben seguir precisamente los pasos 1 al 4 descritos anteriormente, no resulta posible acudir al método fit de Keras que convencionalmente se usa para entrenar las Redes Convolucionales. En lugar de esto se usará el método train_on_batch, que permite controlar lo que ocurrirá en cada iteración del entrenamiento. Así, en primer lugar, crearemos dos lotes de 128 imágenes cada uno: uno con imágenes falsas obtenidas con el Generador y otro con imágenes reales (provenientes del set de entrenamiento). Ver figura 10.

```
# train the generator and discriminator by enumerating batches and epochs.
#
def train(g_model, d_model, gan_model, dataset, latent_dim, n_epochs=100, n_batch=1
    bat_per_epo = int(dataset.shape[0] / n_batch)
    half_batch = int(n_batch / 2) #Disc. trained on half batch real and half batch
    # enumerate epochs
    for i in range(n_epochs):
        # enumerate batches
        for j in range(bat_per_epo):
            # Fetch random 'real' images
            X_real, y_real = generate_real_samples(dataset, half_batch)
            # Train the discriminator using real images
            d_loss1, _ = d_model.train_on_batch(X_real, y_real)
            # generate 'fake' images
            X_fake, y_fake = generate_fake_samples(g_model, latent_dim, half_batch)
            # Train the discriminator using fake images
            d_loss2, _ = d_model.train_on_batch(X_fake, y_fake)
            # Generate latent vectors as input for the generator
            X_gan = generate_latent_points(latent_dim, n_batch)
            # Label generated (fake) images as 1 to fool the discriminator
            y_gan = ones((n_batch, 1))
            # Train the generator (via the discriminator's error)
            g_loss = gan_model.train_on_batch(X_gan, y_gan)
            # Report disc. and gen Losses:
            print('Epoch>%d, %d/%d, d1=%.3f, d2=%.3f g=%.3f' %
                  (i+1, j+1, bat_per_epo, d_loss1, d_loss2, g_loss))
        # evaluate the model performance, sometimes
        if (i+1) % 10 == 0:
            summarize_performance(i, g_model, d_model, dataset, latent_dim)
#########################################
```

Figura 10. Fragmento de código entrenamiento de la GAN. Elaboración propia.

III. Pruebas y resultados

Se propuso en la etapa de prueba evaluar los resultados obtenidos por redes multicapa MLP (MultiLayer Perceptron) de diferentes profundidades entrenados con la base de datos TFD cuyos modelos son reportados por Bengio et al. (2013). Dichos valores de error obtenidos en los diferentes clasificadores se indican a continuación:

a) Clasificador CAE-1: Red tipo autoencoder con una capa interna y clasificador simple cuyo error fue del 27.79%

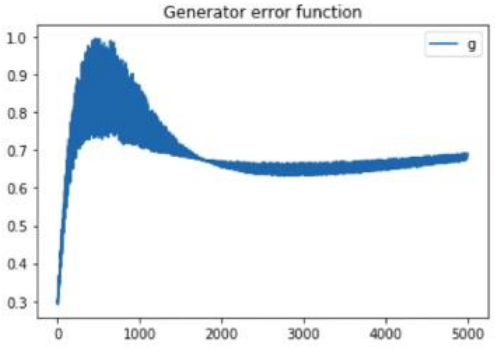
b) Clasificador CAE-2: Es un clasificador de dos capas convolucionales con un valor de error de 23.73%

c) Clasificador DBN-1: Red neuronal con una capa convolucional para la extracción de características relevantes. Con un error del 28.14%

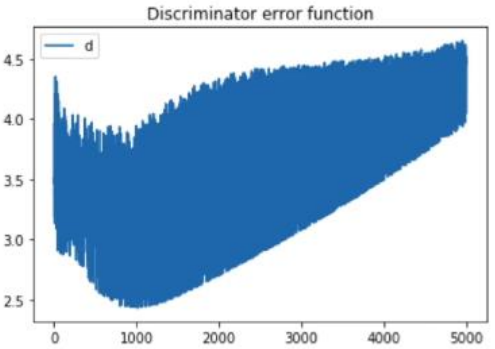
d) Clasificador DBN-2: Se diseñó con una red de 3 capas, se obtuvo un 24.12% de error que incluye mayor número de mapas de características con capas ocultas completamente conectadas tanto a las capas ocultas como a las de salida.

Humanidades, Tecnología y Ciencia, del Instituto Politécnico Nacional
Certificado de Reserva de Derechos al Uso Exclusivo del Título: 04-2010-0326124-14000-203 ISSN: 2007-1957

En el caso del modelo propuesto en el presente trabajo se emplearon 500 muestras (25% del total de muestras) para prueba y 5 mil épocas de entrenamiento para obtener los gráficos de la función de error. Por su parte, el generador en la figura 11a comienza con un valor de error elevado lo cual es esperado ya que al inicio los vectores latentes son ruido, pero con el paso de las épocas de entrenamiento se va estabilizando hacia 0.5. En la figura 11b, el discriminador comienza con valores de error bajo ya que le es fácil identificar las imágenes reales y al final al mejorar el generador le resulta más difícil diferenciar las imágenes falsas de las reales. En la figura 11c, se observan las imágenes obtenidas sintéticamente.



a)



b)

c)

Figura 11. Función de error. a) Generador, b) Discriminador y c) Imágenes Sintéticas obtenidas. Elaboración propia.

Al principio del entrenamiento las imágenes del generador no serán similares a las de un rostro, de modo que para el discriminador será fácil diferenciar las imágenes reales de las falsas (error pequeño). Ver figura 12a. Sin embargo, a medida que avance el entrenamiento, el generador aprenderá poco a poco a producir imágenes parecidas a un rostro humano. Por lo tanto, se logrará confundir al discriminador (error alto), como se observa en la figura 12b. Al final del entrenamiento, la salida del discriminador no será de 0 (imagen falsa), ni 1 (imagen real), sino 0.5. Ver figura 12c.

	Salida del discriminador	
Entrada	**Real (1)**	**Falso (0)**
Imágenes reales (*X*)	**450 TP**	**50 FP**
Imágenes falsas (*X'*)	**50 FN**	**450 TN**

a)

Humanidades, Tecnología y Ciencia, del Instituto Politécnico Nacional
Certificado de Reserva de Derechos al Uso Exclusivo del Título: 04-2010-0326124-14000-203 ISSN: 2007-1957

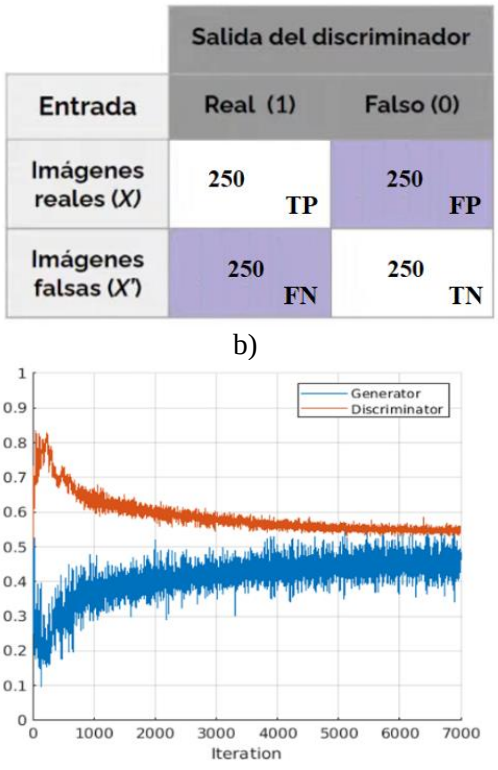


Figura 12. Matriz de confusión. a) Al principio del entrenamiento, b) Al final del entrenamiento y c) Tendencia. Elaboración propia.

Como lo señala Vizcaíno Salazar G. (2002), las métricas de exactitud, sensitividad y especificidad son una medida indirecta de la calidad de los modelos clasificadores. Donde:

$$exactitud = \frac{TP+TN}{TP+TN+FP+FN} \ldots (3)$$

$$sensitividad = \frac{TP}{TP+FN} \ldots (4)$$

$$especificidad = \frac{TN}{TN+FP} \ldots (5)$$

El resumen de las métricas de error obtenidas se presenta en la tabla 1. Se puede observar que en medida del empleo de un mayor número de capas internas en el modelo se alcanza una mayor disminución en el error de clasificación de los rostros. El valor de error menor corresponde al modelo de 4 capas convolucionales propuesto en el presente trabajo alcanzado un valor del 25%.

Tabla 1. Métricas del error en los modelos. Elaboración propia.

MODELO	ERROR (%)	CAPAS
CAE-1	27.79	1
CAE-2	23.73	2
DBN-1	28.14	1
DBN-2	24.12	3
GAN	25.0	4

IV. Conclusiones

Se pudo observar una propuesta para estimar modelos generativos, en la que de forma simultánea dos modelos: uno generativo G, que captura la distribución de datos y un modelo discriminativo D, estima la probabilidad de que una muestra provenga de los datos de entrenamiento, en vez de G. El procedimiento de entrenamiento para G es maximizar la probabilidad de que D cometa un error. Se propusieron arquitecturas convolucionales en los que el sistema se puede entrenar con retropropagación. Sin necesidad del uso de cadenas de Markov. Se mostró el uso de la interpolación lineal, así como el uso de la aritmética de los vectores del espacio latente que fueron capaces de proporcionar el conjunto de muestras falsas para el entrenamiento de la red generadora. En el caso bajo estudio se logró observar cómo al tomar un set de datos que contiene rostros humanos reales, y tras el entrenamiento resultó posible ver que el Generador aprende esta distribución

y es capaz de producir rostros artificiales a partir de una entrada que, en este caso, era simplemente un arreglo con números aleatorios. Adicionalmente, cabe resaltar que la pruebas revelaron que el modelo alcanza menores valores de error en cuanto mayor sea el número de capas que lo conforman. Para el caso particular del presente trabajo se obtuvo un valor del 25% que resultó menor a los modelos conocidos cuyo error oscila entre 23 a 28 %.

V. Referencias

Base de Datos utilizada: UTKFace de Kaggle, https://susanqq.github.io.UTKFace/. (Última consulta 20-03-2024).

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... & Bengio, Y. (2014). Generative adversarial nets. In Advances in neural information processing systems (pp. 2672-2680).

Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434.

Kingma, D. P. and Welling, M. (2014). Auto-encoding variational bayes. In Proceedings of the International Conference on Learning Representations (ICLR).

Bengio, Y., Mesnil, G., Dauphin, Y., and Rifai, S. (2013a). Better mixing via deep representations. In ICML'13.

Vizcaíno Salazar G. (2002). Sensibilidad y Especificidad. Medicina basada en la evidencia y análisis de diseños de investigación clínica. Maracaibo, Venezuela: EDILUZ; 57-71.