Tarea N° 4:
Aprendizaje Profundo Avanzado

Luis Albandoz González

COM4402 – Introducción a Inteligencia Artificial

Escuela de Ingeniería, Universidad de O'Higgins

2 de diciembre, 2023

***Resumen*— Se seleccionan 3 de los 4 códigos para explorar el uso de los diferentes métodos de Aprendizaje Profundo Avanzado.

El primer código elegido se basa en el entrenamiento y en analizar un método de comprensión y reconstrucción de imágenes basado en Autoencoders. Por su parte, el segundo código elegido trata de entrenar y examinar el método de generación de imágenes falsas basados en la arquitectura GAN. Y en el tercer código elegido se ejercita y estudia el método de transferencia de estilos en imágenes basado en la red CNN.

Posteriormente se ejecutan los códigos por separado y se conoce la arquitectura utilizada por cada código en el que será analizado.

Palabras claves— Aprendizaje Profundo Avanzado, Autoencoders, generación de imágenes falsas, GAN, reconstrucción de imágenes, CNN.

I. INTRODUCCIÓN

El Aprendizaje Profundo o Deep Learning es un subcampo del aprendizaje automático (Machine Learning) que utiliza Redes Neuronales Artificiales con múltiples capas para realizar tareas de aprendizaje y reconocimiento de patrones. Estas redes con capaces de aprender representaciones jerárquicas de datos, los que hace esencialmente efectivas para tareas complejas como el reconocimiento de voz, la visión de la computadora y el procesamiento del lenguaje natural.

Por su parte, el Aprendizaje Profundo Avanzado se basa en el uso de técnicas más complejas, incluyendo el uso de redes más sofisticadas, algoritmos de optimización mejorados, técnicas de regularización avanzadas, entre otras técnicas.

II. MARCO TEÓRICO

El objetivo de un Autoencoder según Bandyopadhyay (2021) es "aprender una representación (codificación) de dimensiones inferiores para datos de dimensiones superiores, normalmente para la reducción de dimensionalidad, entrenando a la red para capturar las partes más importantes de la imagen de entrada" (s.p).

A partir de lo anterior, se puede deducir que un Auto Encoder es un tipo especial de red neuronal que se entrena para copiar su entrada en su salida; dada una imagen un Autoencoder primero codifica la imagen en una representación latente de menor dimensión, y luego decodifica la representación latente de vuelta a una imagen.

Las Redes Generativas Adversarias (GAN) según Brownlee (2019) señala que:

> Son una forma inteligente de entrenar un modelo generativo al enmarcar el problema como un problema de aprendizaje supervisado con dos submodelos: el modelo generador que entrenamos para generar nuevos ejemplos y el modelo discriminador

que intenta clasificar los ejemplos como reales (del dominio) o falso (generado). (s.p)

De este enunciado se concluye que las redes generativas (GAN) enmarca un problema de Aprendizaje Supervisado con dos modelos que se entrenan simultáneamente mediante un proceso adversarial. Un generador aprende a crear imágenes que parecen reales, mientras que un discriminador aprende a distinguir las imágenes reales de las falsas.

Según Díaz (2023) una Transferencia Neuronal de Estilos en imágenes "consiste en a partir de dos imágenes, donde una es el contenido y otra es el estilo, combinarlas para generar la misma imagen que en el contenido, pero con el estilo de la otra" (s.p).

De la aseveración anterior se puede afirmar que una Transferencia Neuronal de Estilos de imágenes es una técnica de optimización utilizada para tomar dos imágenes: una imagen de contenido y una imagen de referencia de estilo (como una obra de arte) y mezclarlas para que la imagen de salida se parezca a la imagen de contenido, pero "*pintada*" con el estilo de la imagen de referencia de estilo.

III. METODOLOGÍAS

En el código de Autoencoders (código 1) se entrena el conjunto de datos Fashion MNIST; cada imagen de este conjunto de datos tiene 28x28 píxeles. Se divide el conjunto de datos en 2 grupos: conjunto de entrenamiento que abarcan 60000 datos y el conjunto de prueba en que hay 10000 datos; en total la base de datos presenta 70000 elementos como se muestra en el algoritmo de la Fig. 1

```
(x_train, _), (x_test, _) = fashion_mnist.load_data()

x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.

print (x_train.shape)
print (x_test.shape)
```

Fig. 1 Algoritmo de clasificación conjunto de datos

El siguiente paso se crea una arquitectura que define la función "*Autoencoder*" con dos capas densas: un Encoder que comprime las imágenes en un vector latente de 64 dimensiones, y un Decoder, que reconstruye la imagen original a partir del espacio latente, tal cual se muestra en la Fig. 2

```
class Autoencoder(Model):
  def __init__(self, latent_dim, shape):
    super(Autoencoder, self).__init__()
    self.latent_dim = latent_dim
    self.shape = shape
    self.encoder = tf.keras.Sequential([
      layers.Flatten(),
      layers.Dense(latent_dim, activation='relu'),
    ])
    self.decoder = tf.keras.Sequential([
      layers.Dense(tf.math.reduce_prod(shape), activation='sigmoid'),
      layers.Reshape(shape)
    ])

  def call(self, x):
    encoded = self.encoder(x)
    decoded = self.decoder(encoded)
    return decoded


shape = x_test.shape[1:]
latent_dim = 64
autoencoder = Autoencoder(latent_dim, shape)
```

Fig. 2 Arquitectura definida para Autoencoder

Posteriormente se define otro modelo de arquitectura denominado "*Denoise*" que utiliza la función TensorFlow y Keras, indicando que está creando un modelo personalizado. Este modelo se utiliza para realizar una tarea de denoising Autoencoder en imágenes de 28x28 píxeles en escala de grises; lo que elimina el ruido en las imágenes, mostrando en la Fig. 3

```
class Denoise(Model):
  def __init__(self):
    super(Denoise, self).__init__()
    self.encoder = tf.keras.Sequential([
      layers.Input(shape=(28, 28, 1)),
      layers.Conv2D(16, (3, 3), activation='relu', padding='same', strides=2),
      layers.Conv2D(8, (3, 3), activation='relu', padding='same', strides=2)])

    self.decoder = tf.keras.Sequential([
      layers.Conv2DTranspose(8, kernel_size=3, strides=2, activation='relu', padding='same'),
      layers.Conv2DTranspose(16, kernel_size=3, strides=2, activation='relu', padding='same'),
      layers.Conv2D(1, kernel_size=(3, 3), activation='sigmoid', padding='same')])

  def call(self, x):
    encoded = self.encoder(x)
    decoded = self.decoder(encoded)
    return decoded

autoencoder = Denoise()
```

Fig. 3 Arquitectura Denoise

Por una parte, en el segundo código elegido se crea la función "*make_generator_model*" utiliza capas convolucionales (upsampling) para producir una imagen a partir de una semilla (ruido aleatorio). En la Fig. 4 se muestra un pseudocódigo de la función *generator*.

```
def make_generator_model():
    model = tf.keras.Sequential()
    model.add(layers.Dense(7*7*256, use_bias=False, input_shape=(100,)))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())
```

Fig. 4 Arquitectura generator_model() pseudocódigo

En el siguiente paso se utiliza la función "*make_discriminator_model()*" que consiste en un clasificador de imágenes basado en las Redes Neuronales Convolucionales (CNN), utilizando el discriminador (aun no entrenado) para clasificar imágenes generadas como reales (en valores positivos) y falsas (valores negativos), mostrando en la Fig. 5

```
def make_discriminator_model():
    model = tf.keras.Sequential()
    model.add(layers.Conv2D(64, (5, 5), strides=(2, 2), padding='same',
                            input_shape=[28, 28, 1]))
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.3))

    model.add(layers.Conv2D(128, (5, 5), strides=(2, 2), padding='same'))
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.3))

    model.add(layers.Flatten())
    model.add(layers.Dense(1))

    return model
```

Fig. 5 Arquitectura make_discriminator_model()

Con la arquitectura creada se define la función de pérdida del discriminador, consiste en cuantificar la capacidad del discriminador para distinguir las imágenes reales de las falsas.

En el caso de la Transferencia Neuronal de Estilos (tercer código utilizado) se visualiza la entrada utilizando la arquitectura "*load_img (path_to_img)*" que define una función para cargar una imagen limitando su dimensión de hasta 512 pixeles, ilustrando en la Fig. 6

```
def load_img(path_to_img):
  max_dim = 512
  img = tf.io.read_file(path_to_img)
  img = tf.image.decode_image(img, channels=3)
  img = tf.image.convert_image_dtype(img, tf.float32)

  shape = tf.cast(tf.shape(img)[:-1], tf.float32)
  long_dim = max(shape)
  scale = max_dim / long_dim

  new_shape = tf.cast(shape * scale, tf.int32)

  img = tf.image.resize(img, new_shape)
  img = img[tf.newaxis, :]
  return img
```

Fig. 6 Arquitectura load_img()

Con la función "*load_img*" ya creada, en la Fig. 7 se muestra una función simple "*imshow()*" que permite mostrar imágenes.

```
def imshow(image, title=None):
  if len(image.shape) > 3:
    image = tf.squeeze(image, axis=0)

  plt.imshow(image)
  if title:
    plt.title(title)
```

Fig. 7 Arquitectura imshow()

Para poder construir el modelo se utilizan las librerías TensorFlow y Keras que permiten extraer fácilmente los valores de las capas intermedias usando la API funcional de Keras.

La siguiente función que aparece en la Fig. 8 es un modelo VGG19 que devuelve una lista de salidas de capas intermedias:

```python
def vgg_layers(layer_names):
  """ Creates a VGG model that returns a list of intermediate output values."""
  # Load our model. Load pretrained VGG, trained on ImageNet data
  vgg = tf.keras.applications.VGG19(include_top=False, weights='imagenet')
  vgg.trainable = False

  outputs = [vgg.get_layer(name).output for name in layer_names]

  model = tf.keras.Model([vgg.input], outputs)
  return model
```

Fig. 8 Función vgg_layers()

IV. ANÁLISIS DE RESULTADOS OBTENIDOS

Al analizar los resultados de Autoencoders (código 1) se utiliza el conjunto de datos Fashion MNIST que son imágenes en blanco y negro. Se aplica un Autoencoder utilizando el conjunto de entrenamiento "*x_train*" en 10 épocas; al pasar de una época de menor a mayor se logra observar cómo cambia la pérdida (loss) en el conjunto de entrenamiento y validación a medida que el modelo se entrena a lo largo de las épocas. En este caso, se observa que la pérdida en ambos conjuntos disminuye, lo cual es un buen indicador de que el modelo está aprendiendo y generalizando bien.

Con el modelo ya entrenado, se utiliza el conjunto de prueba para codificar y decodificar imágenes, como se muestra en la Fig. 9

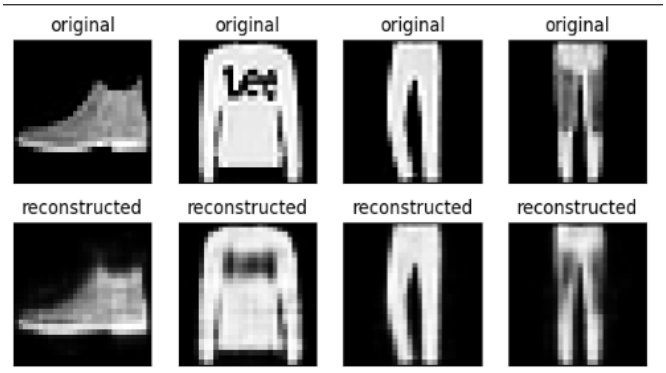


Fig. 9 Ejemplo de imágenes codificadas y decodificadas

También se da el uso del Autoencoder para eliminar ruidos dentro de una imagen, para ello se entrena un Autoencoder con la imagen presentando ruidos como entrada para obtener la imagen original. Se define un nuevo modelo utilizando Autoencoder convolucional usando capas convolucionales *Conv_2D* en el Encoder y capas convolucionales traspuestas *Conv_2D_Transpose* en el Decoder. Al entrenar este modelo con 10 épocas, de una época de menor a mayor se logra ver una disminución en la función de perdida en el conjunto de entrenamiento como en el conjunto de validación.

En la Fig. 10 se muestra el Encoder, donde hay 1320 parámetros en total, los parámetros presentando una capa de convolución bidimensional con una arquitectura simple; y está diseñado para reducir las dimensionalidades; en este caso se reduce la dimensión 28x28 a una dimensión de 7x7.

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 14, 14, 16)        160

 conv2d_1 (Conv2D)           (None, 7, 7, 8)           1160

=================================================================
Total params: 1320 (5.16 KB)
Trainable params: 1320 (5.16 KB)
Non-trainable params: 0 (0.00 Byte)
```

Fig. 10 Modelo Encoder

En caso contrario, el Decoder aumenta la resolución de la imagen. Para este ejemplo aumenta la resolución de 7x7 a 28x28, lo que se obtiene la dimensión original. En el Decoder hay 1897 parámetros totalmente entrenados, por lo que generan imágenes de mayor resolución a partir de características de bajas resoluciones, mostrando en la Fig. 11

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_transpose (Conv2DTr  (None, 14, 14, 8)         584
 anspose)

 conv2d_transpose_1 (Conv2D  (None, 28, 28, 16)        1168
 Transpose)

 conv2d_2 (Conv2D)           (None, 28, 28, 1)         145

=================================================================
Total params: 1897 (7.41 KB)
Trainable params: 1897 (7.41 KB)
Non-trainable params: 0 (0.00 Byte)
```

Fig. 11 Modelo Decoder

A partir de la Fig. 12 se muestran la representación gráfica de las imágenes ruidosas e imágenes sin ruido por el Autoencoder

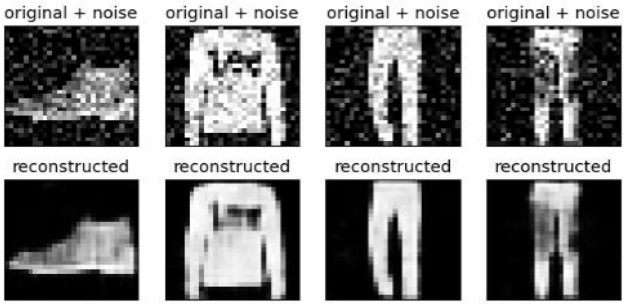
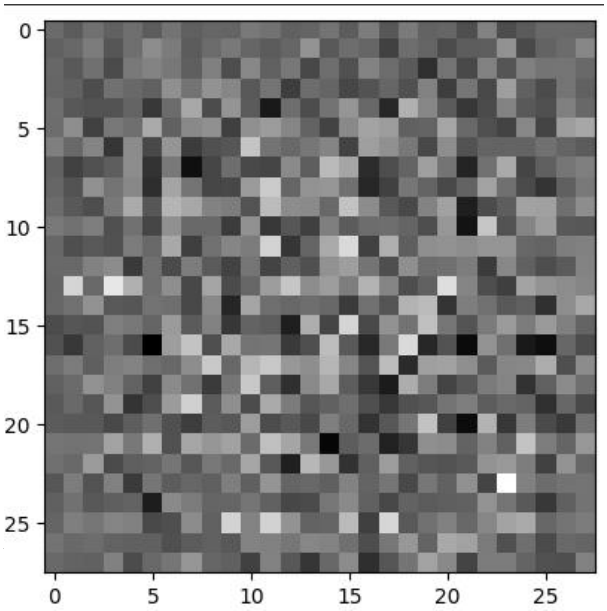


Fig. 12 Ilustración imágenes ruidosas y sin ruido

Al analizar el segundo código elegido (Generative Adversarial Networks (GANs)) se utiliza el conjunto de datos MNIST para entrenar el generador y el discriminador. Las imágenes al comienzo presentan un ruido aleatorio utilizando a través de un generador que utiliza la capa para producir la imagen a partir de una semilla realizando un upsampling varias veces hasta alcanzar el tamaño de imagen deseado de 28x28x1, mostrando en la Fig. 13



Se define un bucle de entrenamiento que comienza con un generador que recibe una semilla aleatoria como entrada durante 50 épocas. A continuación, se utiliza el discriminador que permite clasificar por separado las imágenes reales y las imágenes falsas. Posteriormente se entrena el modelo, por lo que al principio las imágenes presentan un ruido aleatorio, y a medida que avanza el entrenamiento, los dígitos generados parecen cada vez más reales. Cuando se alcanzan el periodo de 50 épocas los datos presentan una mínima cantidad de ruido por lo que se parecen bastante a la base de datos MNIST, ilustrando en la Fig. 14

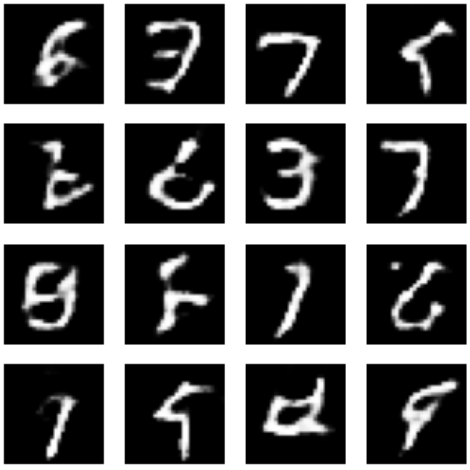


Fig. 14 Dígitos de imágenes

En la Transferencia Neuronal de Estilos (tercer código utilizado) se utilizan dos imágenes para generar una nueva imagen: la imagen de un perro y la composición 7 de Wassily Kandinsky, mostrando en las Fig. 15 y Fig. 16:

Fig. 15 Imagen de un perro

Fig. 16 Composición 7 de Wassily Kandinsky

Se utiliza la Transferencia de Estilos optimizando los contenidos de las imágenes a una nueva imagen con dimensiones particulares; se utiliza el modelo TensorFlow como aparece en la Fig. 17

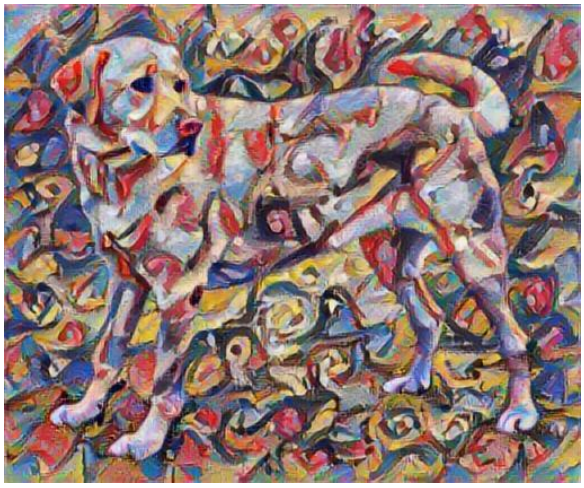


Fig. 17 Modelo de imagen usando TensorFlow

Posterior a ello se construye el modelo VGG 19 usando la API funcional que identifica las entradas y salidas.

El paso siguiente abarca en ejecutar el descenso de la gradiente, con ello se calcula el error cuadrático medio de la salida de la imagen respecto al objetivo. La imagen resultante es un flotante por lo que se define una función "*clip_0_1 (image)*" que permite mantener los valores de los pixeles entre 0 y 1. Luego se optimiza la imagen utilizando una combinación ponderada de dos pérdidas obteniendo la pérdida total. En la Fig. 18 se muestra la imagen actualizada en la que se utiliza la función "*tf.GradientTape*"

Fig. 18 Imagen actualizada

Sin embargo, la desventaja de esta implementación es la producción de varios artefactos de alta frecuencia, lo que genera una pérdida en la variación total de la imagen. En las Fig. 19 y Fig. 20 se muestran las imágenes en las que se usan los componentes de alta frecuencia que es básicamente un detector de bordes, para estos ejemplos se pueden obtener resultados similares a un detector de bordes Sobel.

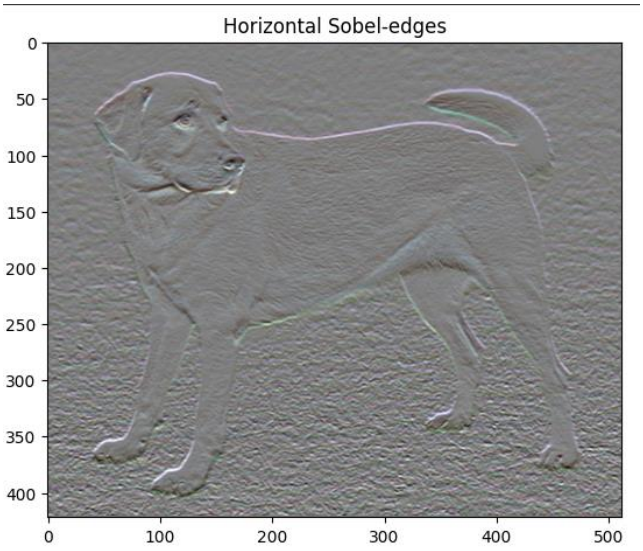


Fig. 19 Borde horizontal

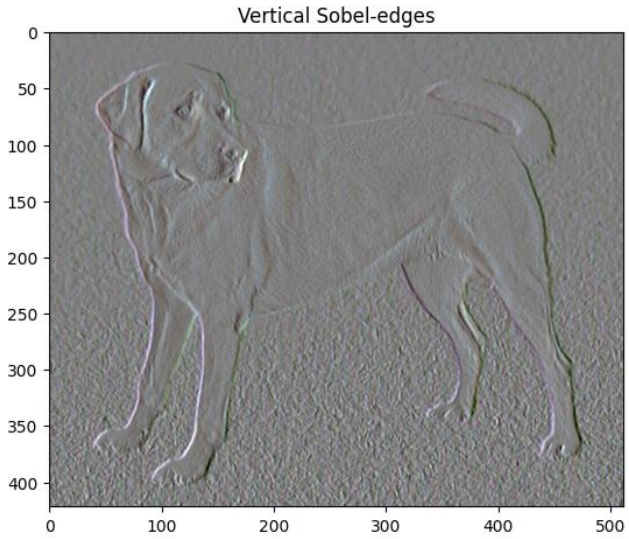


Fig. 20 Borde vertical

Se vuelve a optimizar la imagen, por lo que se reinicia el optimizador junto a la variable de la imagen, en la Fig. 21 se muestra el resultado de la imagen optimizada.

Fig. 21 Imagen optimizada

V. CONCLUSIONES GENERALES

El Aprendizaje Profundo es una de las ramas del Deep Learning que utilizan Redes Neuronales, por lo cual permiten aprender al análisis de grandes cantidades de datos, utilizando técnicas más sofisticadas para resolver problemas complejos.

El Autoencoder es un tipo de Red Neuronal Artificial usado en el Aprendizaje no Supervisado. Está diseñado para aprender representaciones eficientes de datos, generalmente codificando la entrada en un espacio de dimensiones inferiores y luego decodificándola nuevamente a la entrada original. Para este ejemplo se utiliza el conjunto de datos Fashion MNIST presentando dimensiones de 28x28 pixeles, se aplica un Encoder que disminuye las dimensiones a 7x7 pixeles; posterior a ello se aplica un Decoder que logra aumentar las dimensiones a 28x28 pixeles a partir de una dimensión más pequeña.

Las Redes Generativas Adversarias (GAN) son un tipo de aprendizaje profundo que consiste en utilizar Redes Neuronales Convolucionales, un generador y un discriminador que se entrenan de manera adversarial. Se utiliza el conjunto de datos MNIST que al comienzo presentan ruidos en las imágenes; en el siguiente paso que consiste en eliminar ruido de la imagen a través de un entrenamiento con 50 épocas de periodo que permite diferenciar las imágenes reales de las imágenes falsas logrando eliminar el ruido y la imagen tiene una identidad similar al conjunto de datos MNIST.

Por otra parte, la Transferencia Neuronal de Estilos es una técnica de procesamiento de imágenes basada en Redes Neuronales que combina el contenido de una imagen de referencia con el estilo artístico de otra imagen. Esta técnica utiliza modelos pre entrenados de redes neuronales convolucionales, comúnmente modelos de Redes Neuronales Profundas como VGG-19, para lograr este proceso.

En el ejemplo se utilizaron dos imágenes, la primera imagen la de un perro y la segunda imagen trata de la Composición 7 de Wassily Kandinsky en el cual se utiliza la Red Neuronal VGG 19 en el que se combinan ambas imágenes hasta obtener una imagen resulta

BIBLIOGRAFÍA

[1] Bandyopadhyay, H. (2021). Autoencoders in Deep Learning: Tutorial & Use Cases. v7labs. https://www.v7labs.com/blog/autoencoders-guide#h1

[2] Brownlee, J. (2019). A Gentle Introduction to Generative Adversarial Networks (GANs). Machine Learning Mastery. https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans/

[3] Díaz, R. (2023). Domina Neural Style Transfer – Explicación y ejemplo práctico (Pytorch). The Machine Learners. https://www.themachinelearners.com/neural-style-transfer-en-pytorch/#:~:text=La%20t%C3%A9cnica%20de%20Transferencia%20Neuronal,el%20estilo%20de%20la%20otra