REPORTE DE PRÁCTICA

Seminario De Sistemas Basados En Conocimiento

**Nombre Del Alumno** **Código Fecha**

|  |  |  |
| --- | --- | --- |
| Christopher Ceballos Jimenez | 219750442 | 25/Noviembre/2024 |

**NOMBRE DE LA PRÁCTICA/ NUMERO DE LA PRÁCTICA**

|  |
| --- |
| Práctica 9. Implementaron de autoencoder usando MNIST. |

**MARCO TEÓRICO**

|  |
| --- |
| Los autoencoders son un tipo de redes neuronales ampliamente empleadas en Deep learning, destacando en la reducción de ruido en imágenes, como el ruido sal y pimienta. Este tipo de ruido se caracteriza por la presencia aleatoria de píxeles blancos y negros dispersos en la imagen, imitando la apariencia de sal y pimienta esparcida. Los autoencoders funcionan codificando la imagen en una representación de menor tamaño, para luego decodificarla y reconstruir la imagen original, pero con el ruido eliminado. A lo largo de este proceso, la red aprende a atenuar las variaciones aleatorias de píxeles causadas por el ruido, concentrándose en las características principales de la imagen.  La arquitectura estándar de un autoencoder consta de dos componente: el codificador y el decodificador. El codificador reduce la dimensión de la entrada y la transforma en un conjunto de características, mientras que el decodificador reconstruye la entrada original a partir de estas características. Al entrenar un autoencoder con imágenes libres de ruido, la red desarrolla la capacidad de identificar y preservar las características esenciales, lo que le permite luego eliminar el ruido de nuevas imágenes al no considerarlo relevante. |

**IMPLEMENTACIÓN (CÓDIGO)**

|  |
| --- |
| # %%  import matplotlib.pyplot as plt  import numpy as np  import pandas as pd  import tensorflow as tf  from sklearn.metrics import accuracy\_score, precision\_score, recall\_score  from sklearn.model\_selection import train\_test\_split  from tensorflow.keras import layers, losses  from tensorflow.keras.datasets import mnist  from tensorflow.keras.models import Model  # %% [markdown]  # # Resultado aproximado de nuestro Autoencoder  # %% [markdown]  # ![image.png](attachment:image.png)  # %%  #Cargar el repositorio de MNIST con los numeros escritos a mano  (x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()  # %%  x\_train = x\_train.astype('float32') / 255.  x\_test = x\_test.astype('float32') / 255.  x\_train = x\_train[..., tf.newaxis]  x\_test = x\_test[..., tf.newaxis]  print(x\_train.shape)  """  Convierte los valores de los píxeles en las imágenes (originalmente en un rango de 0 a 255) a un rango de 0 a 1 dividiendo por 255.  Además, cambia el tipo de datos a float32, lo cual es más adecuado para cálculos numéricos en redes neuronales  n\_samples: número de imágenes de entrenamiento.  height: altura de cada imagen (p. ej., 28 para MNIST).  width: ancho de cada imagen (p. ej., 28 para MNIST).  1: canal de color (escala de grises).  """  # %% [markdown]  # ## Fn para agregar ruido sal y pimienta  # %%  def add\_salt\_n\_pepper(img, noise\_factor):  """  Agrega ruido de sal y pimienta a una imagen.  El ruido de sal y pimienta reemplaza aleatoriamente los valores de algunos  píxeles en una imagen con valores de intensidad máxima (sal) o mínima (pimienta),  simulando interferencia. Esta función es útil para aumentar datos y mejorar  la robustez de los modelos ante ruido.  Parámetros:  ----------  img : tf.Tensor  Imagen de entrada representada como un tensor. Normalmente debe estar  en un rango [0, 1].  noise\_factor : float  Proporción de ruido a agregar. Valores más altos generan más ruido, y  valores cercanos a 0 dejan la imagen casi sin cambios.  Retorna:  -------  tf.Tensor  La imagen resultante con ruido de sal y pimienta aplicado.  """  img\_out = tf.identity(img)    treshold = 1 - noise\_factor  noise = tf.random.uniform(shape=tf.shape(img), minval=0., maxval=1.)    #sal  img\_out = tf.where(noise < noise\_factor, 1.0, img\_out)    #pimienta  img\_out = tf.where(noise > treshold, 0.0, img\_out)    return img\_out  # %%  noise\_factor = 0.05  x\_train\_noisy = add\_salt\_n\_pepper(x\_train, noise\_factor)  x\_test\_noisy = add\_salt\_n\_pepper(x\_test, noise\_factor)  x\_train\_noisy = tf.clip\_by\_value(x\_train\_noisy, clip\_value\_min=0., clip\_value\_max=1.)  x\_test\_noisy = tf.clip\_by\_value(x\_test\_noisy, clip\_value\_min=0., clip\_value\_max=1.)  """  aplica ruido de sal y pimienta a los conjuntos de datos de entrenamiento (x\_train) y prueba (x\_test)  y luego asegura que los valores de los píxeles estén dentro del rango permitido ([0, 1]).  """  # %% [markdown]  # ### Look at imgs with noise  # %%  n = 10  plt.figure(figsize=(20, 2))  for i in range(n):  ax = plt.subplot(1, n, i + 1)  plt.title("original + noise")  plt.imshow(tf.squeeze(x\_test\_noisy[i]))  plt.gray()  plt.show()  # %% [markdown]  # ### We define a convolutional autoencoder  # %%  class Denoise(Model):  def \_\_init\_\_(self):  super(Denoise, self).\_\_init\_\_()      # encoding part  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)])    # decoding part  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()  # %%  """  Modelo de autoencoder para eliminar ruido de imágenes.  Este modelo está diseñado como un autoencoder convolucional. Consiste en dos partes:  un codificador (encoder) para comprimir las imágenes en una representación latente de menor dimensión  y un decodificador (decoder) para reconstruir las imágenes originales desde la representación comprimida.  El modelo es útil para tareas de eliminación de ruido en imágenes (denoising),  donde se entrena utilizando imágenes ruidosas como entrada y las imágenes originales  como etiquetas de salida.  Métodos:  -------  \_\_init\_\_():  Inicializa las capas del modelo, definiendo el codificador y el decodificador.  call(x):  Define el flujo de datos a través del modelo. Toma una imagen de entrada, la procesa  a través del codificador para obtener una representación latente comprimida y luego  pasa esta representación al decodificador para reconstruir la imagen.  Atributos:  ----------  encoder : tf.keras.Sequential  Parte del modelo encargada de comprimir la imagen de entrada.  Incluye:  - Una capa de entrada con forma (28, 28, 1).  - Dos capas Conv2D con activación ReLU, stride 2 y padding 'same'.  decoder : tf.keras.Sequential  Parte del modelo encargada de reconstruir la imagen a partir de la representación comprimida.  Incluye:  - Dos capas Conv2DTranspose con activación ReLU, stride 2 y padding 'same'.  - Una capa final Conv2D con activación sigmoide para generar la salida reconstruida.  Parámetros:  ----------  x : tf.Tensor  Imagen de entrada con dimensiones (batch\_size, 28, 28, 1).  Retorna:  -------  tf.Tensor  Imagen reconstruida con la misma forma que la entrada (batch\_size, 28, 28, 1).  """  # %%  autoencoder.compile(optimizer='adam', loss=losses.MeanSquaredError())  # %%  # 3 epocas porque se quema mi compu  autoencoder.fit(x\_train\_noisy, x\_train,  epochs=3,  shuffle=True,  validation\_data=(x\_test\_noisy, x\_test))  # %%  encoded\_imgs = autoencoder.encoder(x\_test\_noisy).numpy()  decoded\_imgs = autoencoder.decoder(encoded\_imgs).numpy()  # %%  n = 10  plt.figure(figsize=(20, 4))  for i in range(n):  # display original + noise  ax = plt.subplot(2, n, i + 1)  plt.title("original + salt and pepper")  plt.imshow(tf.squeeze(x\_test\_noisy[i]))  plt.gray()  ax.get\_xaxis().set\_visible(False)  ax.get\_yaxis().set\_visible(False)      bx = plt.subplot(2, n, i + n + 1)  plt.title("reconstructed")  plt.imshow(tf.squeeze(decoded\_imgs[i]))  plt.gray()  bx.get\_xaxis().set\_visible(False)  bx.get\_yaxis().set\_visible(False)  plt.show() |

**RESULTADOS/CONCLUSIONES**

|  |
| --- |
| El proceso de creación y uso de los autoencoders demuestran ser una herramienta eficaz para la edición de imágenes y la eliminación de ruido en éstas. Permiten recrear las imágenes con una fidelidad impresionante incluso cuando ha sido corrupta. |