**Link GIT HUB**

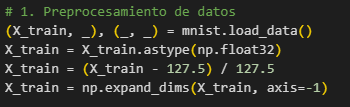
**https://colab.research.google.com/drive/1bn1ISeUWoWE5cSwGUcugN5l\_IDLypOiN#scrollTo=Gj9L0eO2gZnm**

**Informe REDES GAN**

**1. Preprocesamiento de datos:**

El dataset MNIST se compone de imágenes de dígitos escritos a mano con dimensiones 28x28. En la etapa de preprocesamiento, estas imágenes se normalizan para que sus valores estén en el rango [-1, 1]. La normalización ayuda a que la red neuronal convolucional tenga un mejor rendimiento al trabajar con valores más pequeños.

Código de preprocesamiento:



**Generador**: El objetivo del generador es crear imágenes que parezcan reales. Comienza con una entrada aleatoria (ruido) y produce una imagen de salida. Utiliza capas densas, de redimensionamiento y convolucionales para este proceso.

Código del generador:

Texto

Descripción generada automáticamente

Explicación de cada línea:

1. **model = Sequential()**: Aquí se crea un modelo secuencial que permite construir una red neuronal capa por capa en orden secuencial.
2. **model.add(Dense(128 \* 7 \* 7, input\_dim=100))**: Esta línea agrega una capa densa (totalmente conectada) con 6272 neuronas (128 \* 7 \* 7). El número de neuronas se selecciona para que coincida con las dimensiones de la capa de entrada del generador. La capa de entrada tiene una dimensión de 100, por lo que se utiliza **input\_dim=100**. Esta capa toma un vector de ruido de 100 dimensiones como entrada y lo transforma en un tensor de tamaño (128, 7, 7).
3. **model.add(Activation('relu'))**: Aquí se agrega una función de activación ReLU (Rectified Linear Unit) a la salida de la capa densa. ReLU es una función de activación comúnmente utilizada en capas ocultas de redes neuronales debido a su capacidad para aprender representaciones no lineales y evitar el problema de desvanecimiento del gradiente.
4. **model.add(Reshape((7, 7, 128)))**: La capa anterior produce una salida plana, y esta línea cambia la forma de esa salida a un tensor tridimensional (7, 7, 128). Esto se hace para que pueda ser procesado por capas de convolución posteriores.
5. **model.add(UpSampling2D(size=(2, 2)))**: Esta capa realiza una operación de upsampling (aumento de la resolución) del tensor. Duplica las dimensiones de altura y ancho del tensor (en este caso, de 7x7 a 14x14) utilizando interpolación. Esto ayuda a aumentar la resolución de la imagen generada.
6. **model.add(Conv2D(128, kernel\_size=3, padding='same'))**: Agrega una capa de convolución con 128 filtros y un tamaño de kernel de 3x3. El argumento **padding='same'** asegura que la salida tenga las mismas dimensiones que la entrada después de la convolución. Esta capa busca aprender características más detalladas de la imagen.
7. **model.add(Activation('relu'))**: Se agrega otra capa ReLU después de la convolución para introducir no linealidad en la red y permitir que el generador aprenda representaciones más complejas.
8. **model.add(UpSampling2D(size=(2, 2)))**: Otra capa de upsampling para aumentar aún más la resolución de la imagen (de 14x14 a 28x28).
9. **model.add(Conv2D(1, kernel\_size=3, padding='same'))**: La última capa de convolución tiene un solo filtro y un tamaño de kernel de 3x3. Esto produce una única imagen generada. La función 'tanh' se utiliza como función de activación en esta capa.
10. **model.add(Activation('tanh'))**: Se utiliza 'tanh' (tangente hiperbólica) como función de activación en la capa de salida. 'tanh' es apropiada aquí porque produce valores en el rango [-1, 1], que es el rango de valores de los píxeles en las imágenes originales escaladas.

Discriminador:

Su función es decidir si una imagen es real o ha sido generada. Utiliza capas convolucionales.

Código del discriminador:

Texto

Descripción generada automáticamente

Explicación de cada línea

1. **model = Sequential()**: Al igual que en el generador, aquí se crea un modelo secuencial para el discriminador.
2. **model.add(Conv2D(64, kernel\_size=3, strides=2, input\_shape=(28, 28, 1), padding="same"))**: Se agrega una capa de convolución con 64 filtros y un tamaño de kernel de 3x3. Los argumentos **strides=2** especifican que la capa utiliza un desplazamiento de 2 píxeles en cada dirección durante la convolución, lo que reduce a la mitad las dimensiones espaciales de la salida. **input\_shape=(28, 28, 1)** establece las dimensiones de entrada de la imagen en 28x28 píxeles en escala de grises, y **padding="same"** garantiza que la salida tenga las mismas dimensiones que la entrada.
3. **model.add(Activation('relu'))**: Se agrega una función de activación ReLU después de la capa de convolución. Se utilizada para aprender características no lineales en la detección de características de la imagen.
4. **model.add(Flatten())**: Esta capa aplana la salida de la capa de convolución anterior para convertirla en un vector unidimensional. Esto se hace para que la salida de la capa de convolución pueda ser alimentada a capas densas.
5. **model.add(Dense(1))**: Se agrega una capa densa (totalmente conectada) con una sola neurona de salida. La neurona única se utiliza para producir una única salida que representa la probabilidad de que la imagen de entrada sea real o falsa.
6. **model.add(Activation('sigmoid'))**: Se utiliza 'sigmoid' como función de activación en la capa de salida. Esto produce una salida en el rango [0, 1], que se interpreta como la probabilidad de que la imagen de entrada sea real (1) o falsa (0).

Detalles de arquitectura y hiperparámetros:

El generador toma una entrada de dimensión 100 y, mediante diversas transformaciones, produce imágenes de 28x28. El discriminador, por otro lado, evalúa estas imágenes y devuelve una probabilidad de que sean reales.

Ambas redes utilizan la función de activación ReLU. El generador finaliza con una función de activación tanh la utilice porque es una elección común para las capas de salida en los generadores de GANs, especialmente cuando se trabaja con imágenes, mientras que el discriminador utiliza una función sigmoide porque devuelve 1 o 0 para decir si la imagen es real o falsa.

Luego se ensambla la GAN utilizando un generador y un discriminador dados. el discriminador se configura para ser no entrenable para asegurar que sólo el generador se actualice durante el entrenamiento.

Texto

Descripción generada automáticamente

Explicación de cada línea

1. **discriminator.compile(optimizer=Adam(0.0002, 0.5), loss='binary\_crossentropy', metrics=['accuracy'])**: Aquí, se compila el discriminador utilizando el optimizador Adam con una tasa de aprendizaje de 0.0002 y un factor de decaimiento. Se utiliza 'binary\_crossentropy' como función de pérdida ya que el discriminador tiene una sola neurona de salida y está realizando una tarea de clasificación binaria (real o falsa). También se especifica que se deben calcular las métricas de precisión durante el entrenamiento para evaluar el rendimiento del discriminador.
2. **z = Input(shape=(100,))**: Se crea una capa de entrada llamada **z** con una forma de (100,). Esto representa el espacio latente, desde el cual se muestrearán los vectores de ruido para generar imágenes falsas.
3. **img = generator(z)**: Aquí, se genera una imagen falsa utilizando el generador. El generador toma un vector de ruido del espacio latente como entrada y produce una imagen generada.
4. **discriminator.trainable = False**: Se establece la propiedad **trainable** del discriminador en **False**. Esto se hace para asegurarse de que durante el entrenamiento de la GAN, solo se actualicen los pesos del generador y no los del discriminador. Esto es fundamental para el proceso de entrenamiento adversarial.
5. **valid = discriminator(img)**: El discriminador se aplica a la imagen generada para obtener una salida que representa la probabilidad de que la imagen sea real o falsa. Esta salida se almacena en la variable **valid**.
6. **gan = Model(z, valid)**: Se crea un nuevo modelo llamado **gan** que toma como entrada un vector de ruido del espacio latente (**z**) y produce la salida **valid** del discriminador. En otras palabras, este modelo compone el generador y el discriminador en una única red que se utilizará para entrenar el generador.
7. **gan.compile(optimizer=Adam(0.0002, 0.5), loss='binary\_crossentropy')**: Se compila el modelo GAN utilizando el mismo optimizador y función de pérdida que el discriminador. La idea aquí es entrenar el generador para engañar al discriminador, por lo que se utiliza la misma función de pérdida 'binary\_crossentropy' para calcular la pérdida del generador.

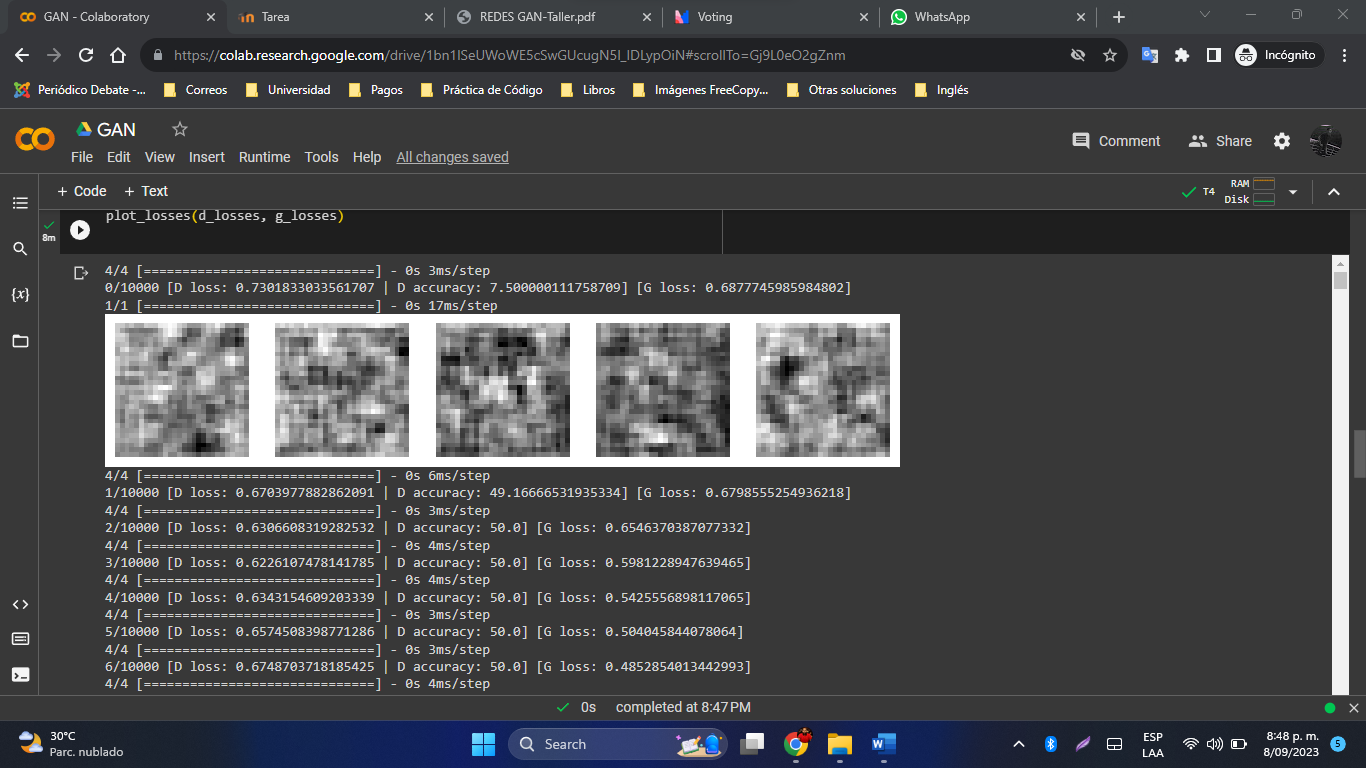
Texto

Descripción generada automáticamente

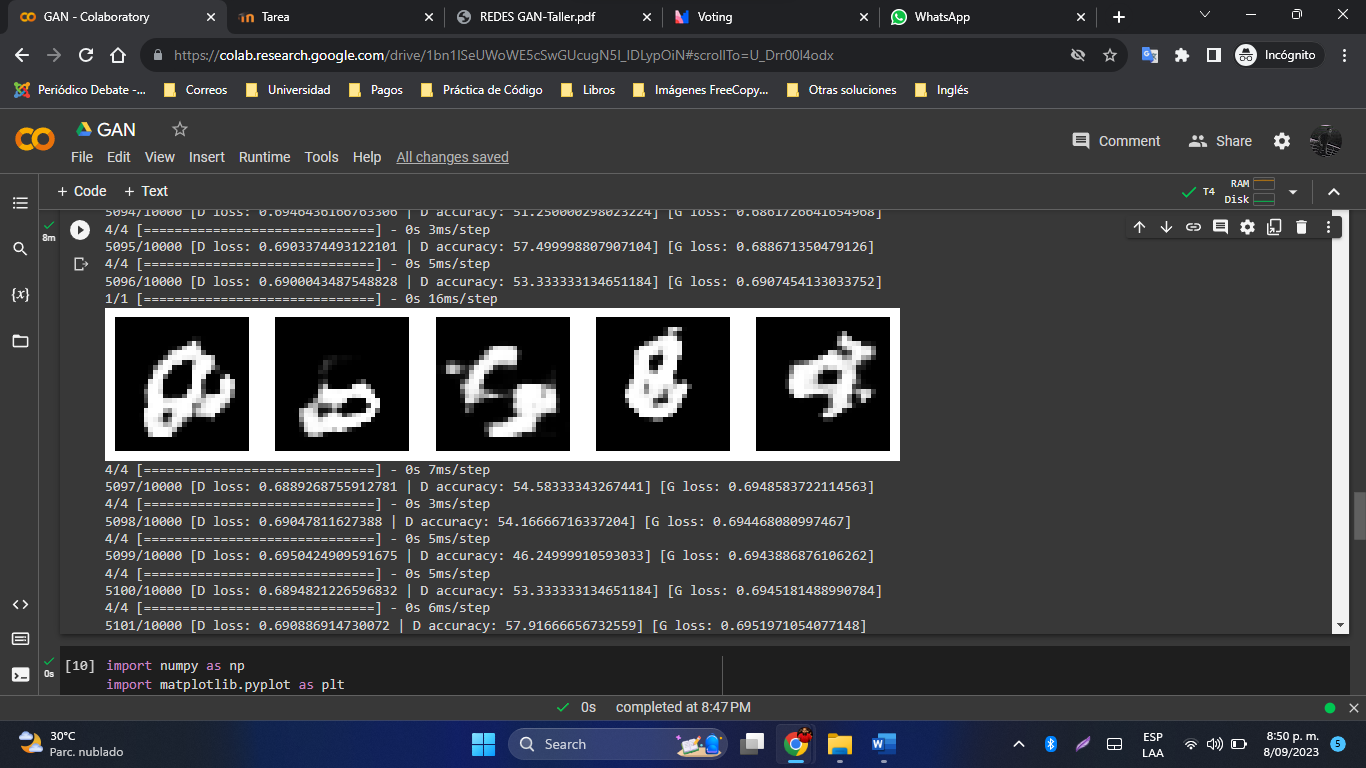
Explicación de cada línea

1. **for epoch in range(epochs):**: Inicia un bucle que se ejecutará durante el número especificado de épocas (epochs). Cada época representa una pasada completa a través de los datos de entrenamiento.
2. **idx = np.random.randint(0, X\_train.shape[0], batch\_size)**: Selecciona de manera aleatoria un conjunto de índices de tamaño **batch\_size** a partir de los datos de entrenamiento **X\_train**. Esto se utiliza para crear lotes de imágenes tanto reales como generadas.
3. **imgs = X\_train[idx]**: Obtiene las imágenes correspondientes a los índices seleccionados en el paso anterior. Estas imágenes son las imágenes reales que se utilizarán para entrenar al discriminador.
4. **noise = np.random.normal(0, 1, (batch\_size, 100))**: Genera ruido aleatorio con una distribución normal (media 0 y desviación estándar 1) para crear el ruido de entrada para el generador. Se genera un lote de ruido con un tamaño de **batch\_size** y 100 dimensiones.
5. **gen\_imgs = generator.predict(noise)**: Utiliza el generador para predecir imágenes a partir del ruido generado en el paso anterior. Estas imágenes generadas se utilizan para entrenar al discriminador.
6. **real\_labels = np.ones((batch\_size, 1))**: Crea etiquetas de "1" para las imágenes reales. Estas etiquetas se utilizan para indicar al discriminador que las imágenes son reales durante su entrenamiento.
7. **fake\_labels = np.zeros((batch\_size, 1))**: Crea etiquetas de "0" para las imágenes generadas. Estas etiquetas se utilizan para indicar al discriminador que las imágenes son falsas durante su entrenamiento.
8. **d\_loss\_real = discriminator.train\_on\_batch(imgs, real\_labels)**: Entrena al discriminador con las imágenes reales y las etiquetas reales. El método **train\_on\_batch** calcula la pérdida y la precisión del discriminador en este lote de imágenes reales.
9. **d\_loss\_fake = discriminator.train\_on\_batch(gen\_imgs, fake\_labels)**: Entrena al discriminador con las imágenes generadas y las etiquetas falsas. Al igual que en el paso anterior, se calcula la pérdida y la precisión del discriminador.
10. **d\_loss = 0.5 \* np.add(d\_loss\_real, d\_loss\_fake)**: Calcula la pérdida total del discriminador como el promedio de la pérdida en las imágenes reales y generadas. Esto se hace para calcular la pérdida total de la GAN de manera más estable.
11. **noise = np.random.normal(0, 1, (batch\_size, 100))**: Se genera otro lote de ruido aleatorio para el generador.
12. **valid\_labels = np.ones((batch\_size, 1))**: Crea etiquetas de "1" para las imágenes generadas. Estas etiquetas se utilizan para indicar al generador que las imágenes generadas son válidas (intentará engañar al discriminador).
13. **g\_loss = gan.train\_on\_batch(noise, valid\_labels)**: Entrena al generador para que genere imágenes que el discriminador considere como válidas. Se calcula la pérdida del generador en este proceso.
14. **print(f"{epoch}/{epochs} [D loss: {d\_loss[0]} | D accuracy: {100 \* d\_loss[1]}] [G loss: {g\_loss}]")**: Imprime el progreso del entrenamiento en cada época. Muestra la época actual, la pérdida y la precisión del discriminador, así como la pérdida del generador.
15. **if epoch % sample\_interval == 0:**: Entra en este bloque de código solo si la época actual es un múltiplo del intervalo de muestra especificado.
16. **sample\_noise = np.random.normal(0, 1, (5, 100))**: Genera un pequeño lote de ruido aleatorio para generar ejemplos de imágenes.
17. **generated\_images = generator.predict(sample\_noise)**: Utiliza el generador para predecir imágenes a partir del ruido generado.
18. **generated\_images = 0.5 \* generated\_images + 0.5**: Reescala las imágenes generadas de modo que estén en el rango [0, 1] en lugar de [-1, 1].
19. **d\_losses.append(d\_loss[0])** y **g\_losses.append(g\_loss)**: Agrega las pérdidas del discriminador y el generador a las listas **d\_losses** y **g\_losses**, respectivamente. Estas listas se utilizan para realizar un seguimiento de las pérdidas a lo largo del entrenamiento.
20. El código restante crea una visualización de las imágenes generadas y las muestra en un formato de 1 fila y 5 columnas en una figura de Matplotlib.

INICIO



MITAD



FINAL

Captura de pantalla de computadora

Descripción generada automáticamente

En conclusión no se logra ver con calidad que números esta creando esto sugiere que la red GAN no quedo bien entrenada por diferentes factores ya sea por más épocas o ajustar la tasa de aprendizaje, uno de los modelos se vuelve demasiado demasiado rápido y puede evitar que el otro modelo aprenda adecuadamente, ya sea la arquitectura, o la inicialización, ya hay que experimentar ajustando hasta poder visualizar mejor la imágenes.

* Inicializa listas para almacenar las pérdidas del generador, el discriminador durante el entrenamiento, y la variable generated\_images para obtener una imagen aleatoria mas adelante.

Texto

Descripción generada automáticamente

* La función se utiliza para visualizar las pérdidas del generador y el discriminador durante el entrenamiento.

Texto

Descripción generada automáticamente

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

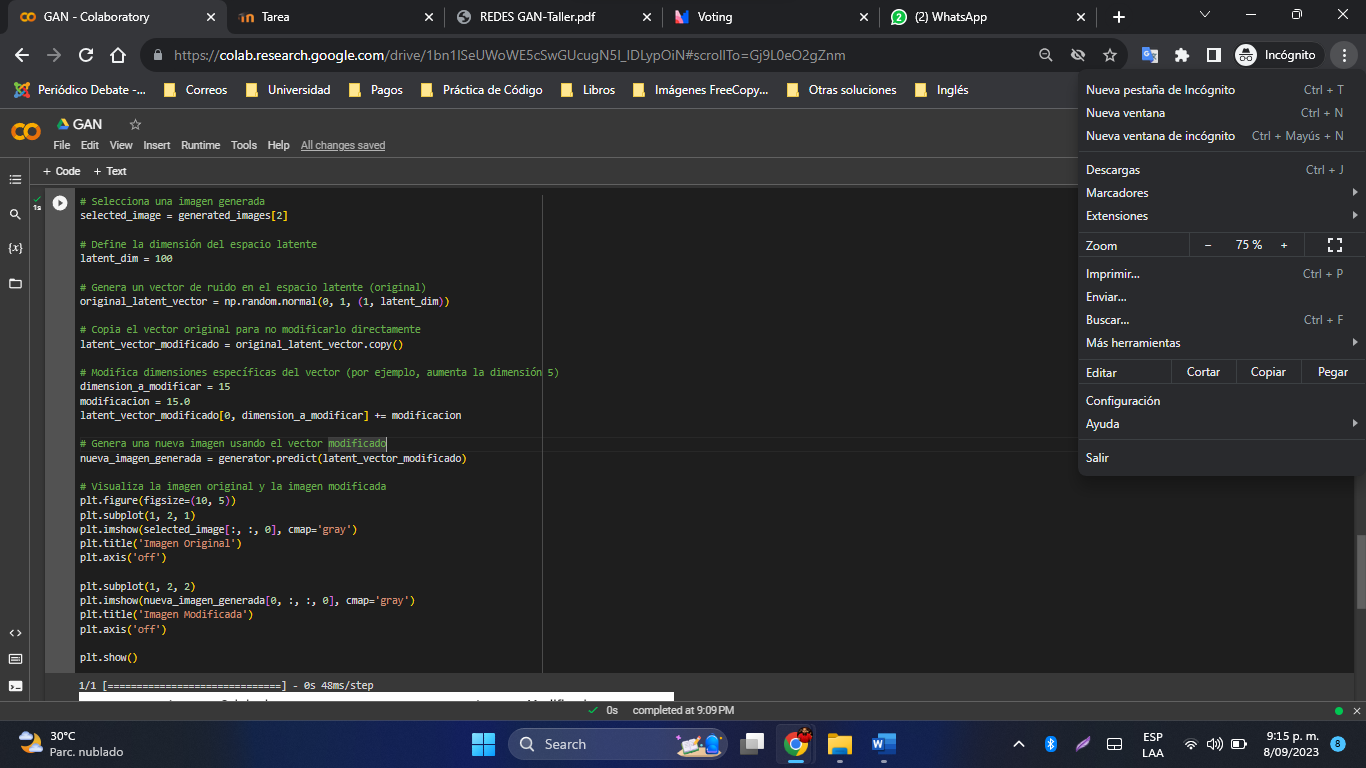
Generador vs. Discriminador: Durante el intervalo maso menos 20 y 30 en el que la pérdida del generador sube y la del discriminador baja, tu discriminador ha "aprendido" mejor que el generador. Específicamente, el discriminador ha mejorado en identificar las imágenes generadas como falsas, mientras que el generador todavía está luchando por crear imágenes que engañen al discriminador.

Equilibrio Posterior: Después de ese intervalo, las pérdidas se estabilicen significa que los modelos han alcanzado un equilibrio. Es posible que el generador haya mejorado ligeramente, lo que lleva al discriminador a no ser tan seguro en sus predicciones. Sin embargo, este equilibrio no necesariamente indica que las imágenes generadas son de alta calidad, sino que el generador y el discriminador han alcanzado un punto donde ninguno supera significativamente al otro.

Ya las ultimas línea de código solo ejecutan las funciones para cada punto que vimos

Texto

Descripción generada automáticamente



Explicación de cada línea

Seleccionar una imagen generada: El código comienza seleccionando una imagen de las imágenes generadas (en este caso, la tercera imagen) para visualizarla posteriormente.

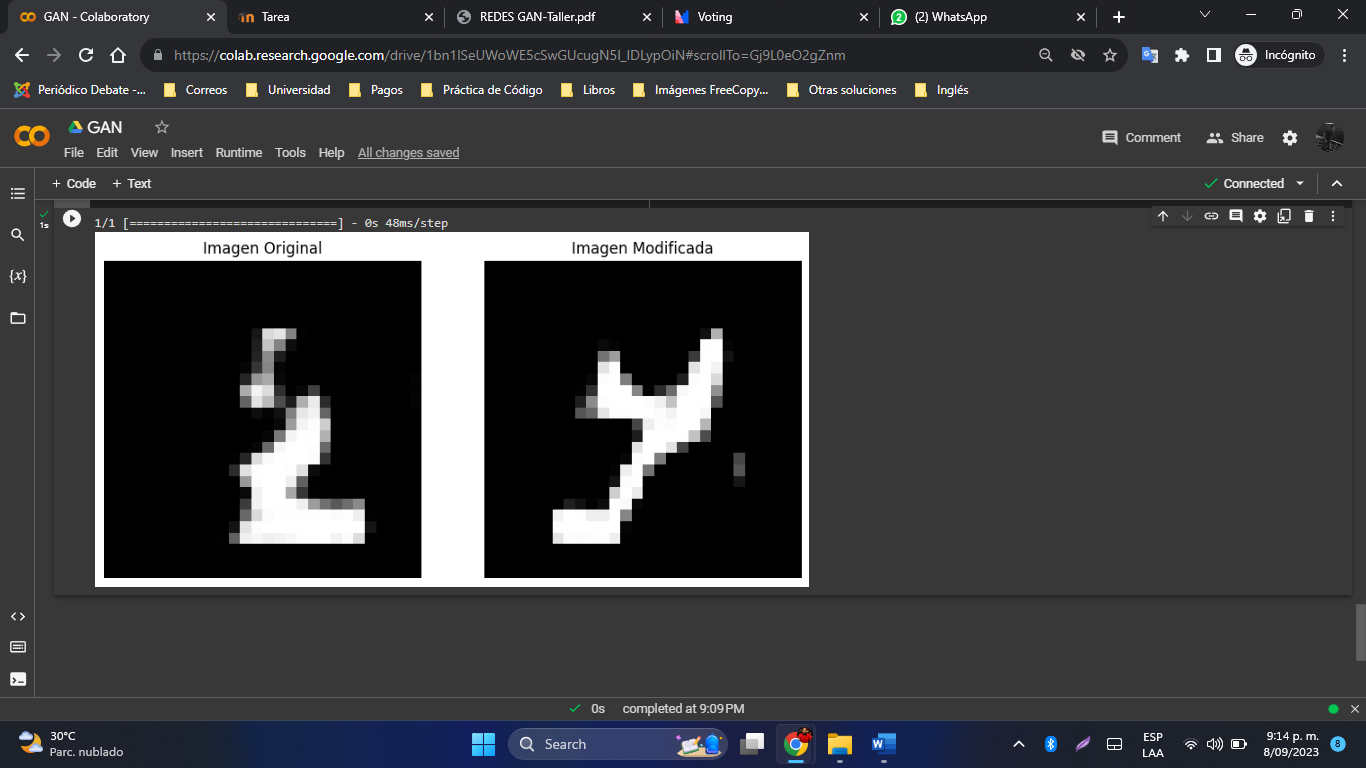
Definir el Espacio Latente: Se establece que la dimensión del espacio latente es de 100. Esto significa que cada imagen generada es el resultado de una entrada (vector de ruido) en este espacio de 100 dimensiones.

Crear Vector de Ruido Original: Se genera un vector de ruido inicial que se utilizará como entrada al generador. Este vector sigue una distribución normal con una media de 0 y una desviación estándar de 1.

Modificación del Vector de Ruido: Se crea una copia del vector original para no modificarlo directamente. Luego, se selecciona una dimensión específica del vector (en este caso, la dimensión 15) y se le añade un valor de modificación (15.0 en este caso). Esta manipulación tiene como objetivo observar cómo ciertas dimensiones en el espacio latente afectan la salida generada.

Generar una Nueva Imagen: Usando el vector modificado como entrada, se genera una nueva imagen a través del modelo generador.

Visualización: Finalmente, el código muestra la imagen original seleccionada y la nueva imagen generada después de la modificación. Esto te permite comparar visualmente las diferencias causadas por la modificación en el espacio latente.



Se evidencia que al modificar se ve que la imagen cambia porque al ajustar valores influimos en alguna dimensión que hace verse de una manera como si fuera un carro una dimensión para la llanta otra para la ventana y al ajustar esta del numero no podemos evidenciar bien en donde se modificar la dimensión.