Instituto Politécnico Nacional

Unidad Profesional Interdisciplinaria

en Ingeniería y Tecnologías

Avanzadas

Multimedia

BMP y Estenografía

Soto Gutiérrez Ian Alexis

3TM2

Como primera parte de la práctica se probó el código proporcionado para verificar que todo funcionara correctamente, para ello se tomó una foto aleatoria en formato BMP para demostrar que se cumpliera con ocultar un mensaje por medio de la técnica LSB en donde se varía el ultimo bit de una cadena de bytes, el fin de hacerlo mediante esta técnica es que las imágenes resultantes sean lo más parecidas a la originales, o sea que los cambios sean prácticamente imperceptibles al ojo humano.

Código cargado:

```
import struct
def leer_bmp(filepath):
 '''Retorna (header_bytes, pixels, width, height, row_size)'''
 with open(filepath, 'rb') as f:
   data = f.read()
   offset = struct.unpack_from('<I', data, 10)[0]
   width = struct.unpack_from('<i', data, 18)[0]
   height = struct.unpack_from('<i', data, 22)[0]
   row_size = (width * 3 + 3) & ~3
   header = bytearray(data[:offset])
   pixels = bytearray(data[offset:])
 return header, pixels, width, height, row_size

def guardar_bmp(filepath, header, pixels):
   with open(filepath, 'wb') as f:
    f.write(header)
    f.write(pixels)

def embed_lsb(src_path, dst_path, mensaje):
 header, pixels, width, height, row_size = leer_bmp(src_path)
 msg_bytes = mensaje.encode('utf-8')
```

```
 msg_len = len(msg_bytes)
# Convertir longitud (4 bytes) + mensaje a flujo de bits
 datos= struct.pack('<I', msg_len) + msg_bytes
 bits = []
 for byte in datos:
   for i in range(7, -1, -1): # MSB primero
     bits.append((byte >> i) & 1)
# Verificar capacidad
     capacidad = (len(pixels) * 3) // 3 * 3 # múltiplos de 3 (B,G,R porpíxel)

# Simplificado: usar bytes secuenciales (saltando padding no es necesarioaquí)
     if len(bits) > len(pixels):
      raise ValueError('Mensaje demasiado largo para esta imagen')
# Incrustar bits en LSB de cada byte de canal
     pixels_mod = bytearray(pixels)
     for idx, bit in enumerate(bits):
        pixels_mod[idx] = (pixels_mod[idx] & 0xFE) | bit # limpiar LSB e insertar bit
     guardar_bmp(dst_path, header, pixels_mod)
     print(f'[OK] Mensaje de {msg_len} bytes incrustado en {dst_path}')

def extract_lsb(stego_path):
  _, pixels, _, _, _ = leer_bmp(stego_path)

  # 1. Leer primeros 32 bits para obtener la longitud del mensaje
  len_bits = [pixels[i] & 1 for i in range(32)]

  # Reconstruir los 4 bytes del mensaje a partir de los bits extraídos
  msg_len_bytes = bytearray(4)
  for byte_idx in range(4):
```

```
        current_byte = 0
        for bit_offset in range(8):
            current_byte = (current_byte << 1) | len_bits[byte_idx * 8 + bit_offset]
        msg_len_bytes[byte_idx] = current_byte

    # Desempaquetar la longitud del mensaje (little-endian, '<I')
    msg_len = struct.unpack('<I', msg_len_bytes)[0]

    # 2. Calcular cuántos bits totales debemos leer (32 de longitud + mensaje)
    total_bits = 32 + msg_len * 8
    msg_bits = [pixels[i] & 1 for i in range(32, total_bits)]

    # 3. Reconstruir los bytes a partir de los bits extraídos
    msg_bytes = bytearray()
    for i in range(0, len(msg_bits), 8):
        byte = 0
        for bit in msg_bits[i:i+8]:
            byte = (byte << 1) | bit
        msg_bytes.append(byte)

    return msg_bytes.decode('utf-8')

embed_lsb('kiss.bmp', 'kisstego.bmp', 'TELEMÁTICA SECRETA 2025')
recuperado = extract_lsb('stego.bmp')
print(f'Mensaje recuperado: {recuperado}')
assert recuperado == 'TELEMÁTICA SECRETA 2025', '¡Error en la extracción!'
print('Prueba exitosa.')

import math
```

```
def calcular_psnr(original_path, stego_path):
 _, pix_orig, w, h, rs = leer_bmp(original_path)
 _, pix_steg, _, _, _ = leer_bmp(stego_path)
 mse = sum((a - b)**2 for a, b in zip(pix_orig, pix_steg)) / (w * h * 3)
 if mse == 0:
   return float('inf')
 psnr = 10 * math.log10(255**2 / mse)
 print(f'MSE: {mse:.6f}')
 print(f'PSNR: {psnr:.2f} dB (>40 dB: cambio imperceptible)')
 return psnr
calcular_psnr('kiss.bmp', 'kisstego.bmp')
```

Figura 1. Imagen original para pruebas.

Figura 2. Imagen modificada.

Como se puede observar en las imágenes anteriores, podemos decir a simple vista que son iguales, pero ¿Cómo verificamos que no son así?

Visualizando la salida del código dentro de Google colab:

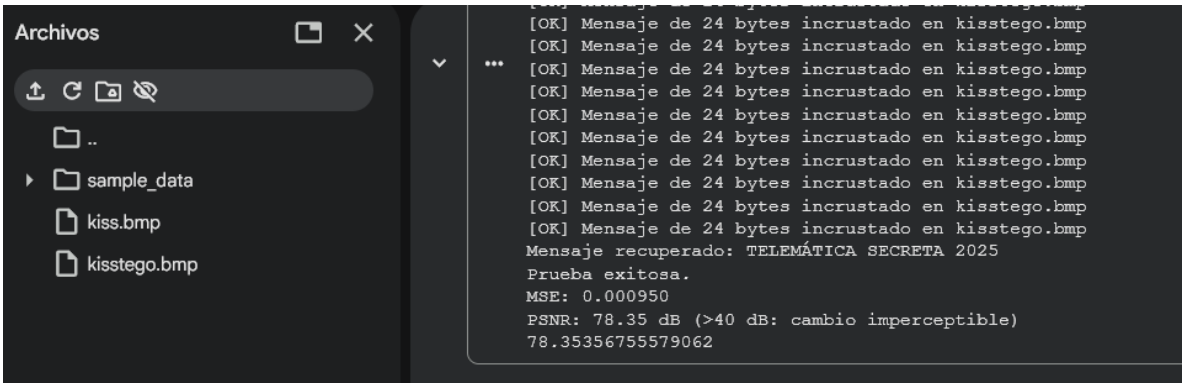

Figura 3. Resultados del código

Podemos observar en la figura 3 que el código funcionó exitosamente y que se realizaron las modificaciones pertinentes dentro de la imagen original, pero ¿Qué significa lo que vemos como salida del script?

- MSE: Se encarga de indicar el error promedio por píxel, al tener un resultado lejano a 1 (0.00095) significa que dicho error es prácticamente inexistente.
- PSNR: Es una función utilizada para cuantificar la calidad al momento de reconstruir una imagen donde se considera que si el resultado es mayor a 40 dB entonces el cambio es imperceptible para el ojo humano, esto porque este es el límite donde la distorsión o el ruido incrustado en la imagen es muy bajo como para ser percibido a simple vista. El resultado que se obtuvo de la imagen de prueba significa que se obtuvo casi el doble de la calidad original (78.32 dB), por ende, todo cambio es imperceptible.
- Mensaje recuperado: Es aquí donde se verifica que realmente funcionó la prueba de esconder un texto dentro de la imagen ya que esta es la manera más sencilla de corroborarlo porque si nos vamos a un analizador de hexadecimal y subimos la imagen veremos texto “basura”.

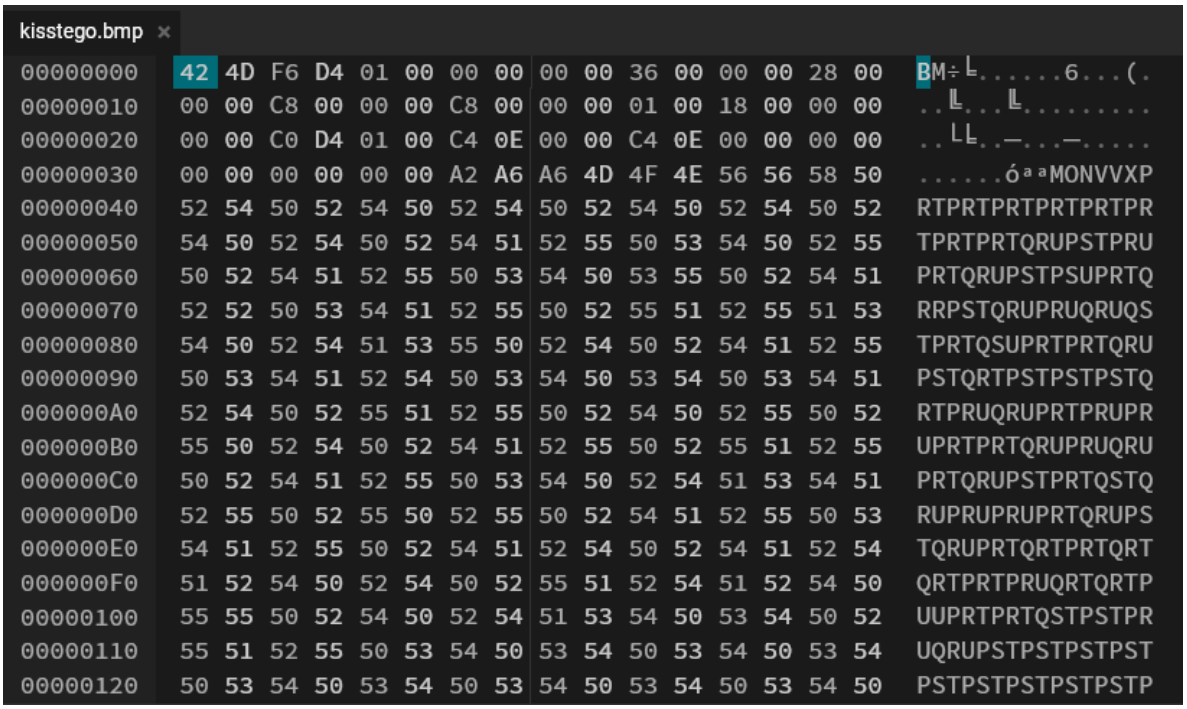

Figura 4. Analizador de hexadecimal.

Como lo observamos en la figura 4, no tenemos el texto oculto mostrado a simple vista, sabemos que primero están lo valores que representan los meta datos de la imagen y después de ello viene el texto oculto pero para encontrarlo tendríamos que analizar byte por byte, sacar su valor en binario y de ahí reconstruir letra a letra hasta completar la palabra lo cual resulta un proceso tedioso, es por esa razón que el estatus de salida del código nos resulta ser la manera más sencilla de corroborar que se cumplió con el objetivo de la esteganografía.

Experimento de capacidad

Realice las siguientes pruebas y complete la tabla de resultados.

Imagen (px)	Tamaño de msj	PSNR obtenido	¿Imperceptible?
200 x 200	50 bytes	75.60 dB	Sí
200 x 200	500 bytes	65.91 dB	Sí
512 x 512	5,000 bytes	64.12 dB	Sí
512 x 512	Capacidad máx.	51.15 dB	Sí

Resultados visuales de la tabla:

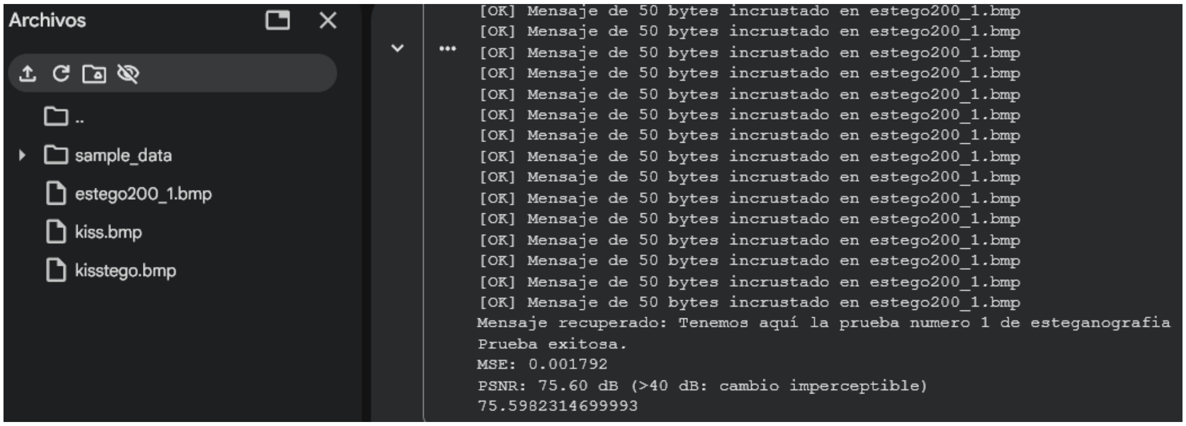

Figura 5. Resultados imagen 200 x 200 con 50 bytes.

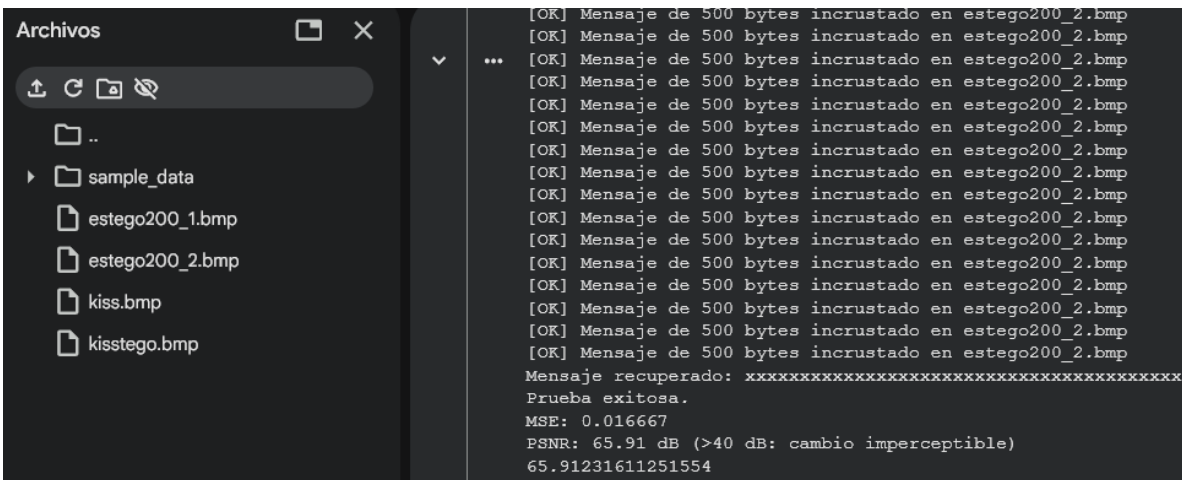

Figura 6. Resultados imagen 200 x 200 con 500 bytes.

Para la obtención de 500 bytes se realizó una función en Python que generara una cadena de texto con un carácter "x" repetido 500 veces.

```
msj_500= "x"*500
embed_lsb('kiss.bmp', 'estego200_2.bmp', msj_500)
recuperado = extract_lsb('estego200_2.bmp') # Cambiado de
print(f'Mensaje recuperado: {recuperado}')
assert recuperado == msj_500, '¡Error en la extracción!'
print('Prueba exitosa.')
```

Figura 7. Función para repetir carácter.

Podemos observar que la variable "*msj_500*" se encarga de repetir el carácter "x" y posteriormente esa es la función que se pasa como parámetro dentro de *embed_lsb* y *recuperado*.

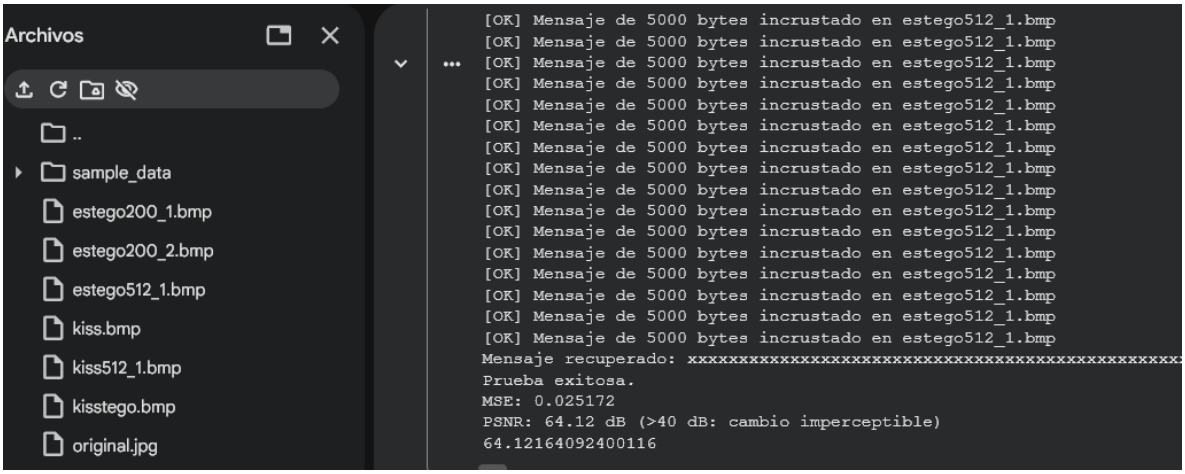

Figura 8. Resultados imagen de 512x512 con 5KB.

Al igual que en la imagen con texto de 500 bytes se utilizó una función que completara los bytes necesarios, en este caso fueron 5000, para corroborarlo podemos copiar el mensaje recuperado y mandarlo a un contador de caracteres:

Figura 9. Comprobación de los 5000 caracteres.

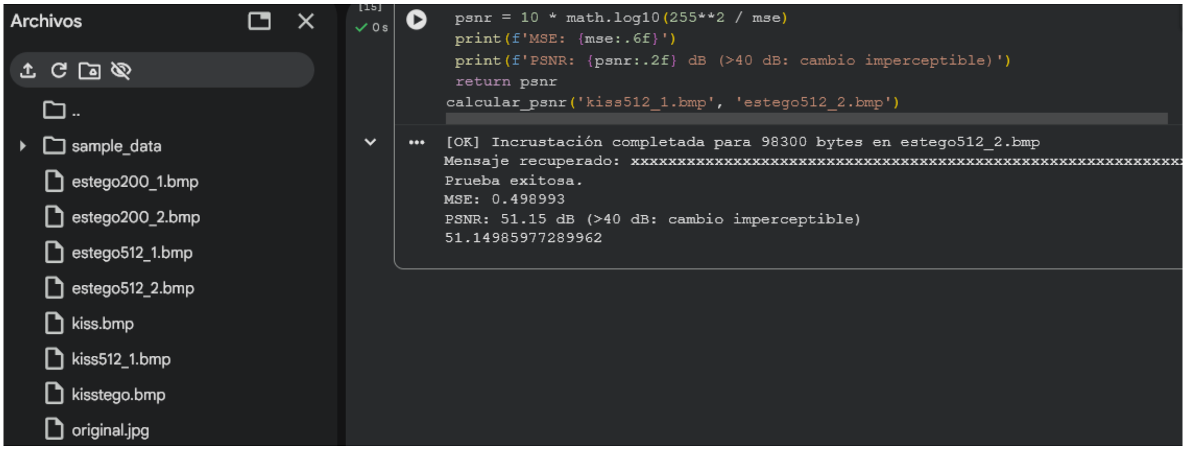

Figura 10. Resultados imagen 512 x 512 con capacidad máx (98300 bytes)

Preguntas de análisis.

1. ¿Qué sucede si se intenta guardar un mensaje más largo que el tamaño de la imagen ? Se obtendría un error de overflow y entraría a una sección de código: *Mensaje demasiado largo para esta imagen*. Para evitar dicho desbordamiento se debe de considerar el tamaño de la imagen desde un inicio, en este caso que tenemos 512x512 en total tenemos 262,144 píxeles y considerando que se tienen 3 canales para colores (RGB) tenemos 262,144x3 obteniendo un valor de 786,432 bytes disponibles dentro de la imagen, dividiendo dicho valor entre los 8 bytes necesarios para ocultar el mensaje tenemos 98,304 bytes, pero debemos restar los 4 bytes iniciales que BMP usa para guardar la longitud, por tanto tenemos un total de 98,300 bytes como capacidad máxima.
2. Compare visualmente la imagen original con la estenografiada. ¿Hay diferencias observadas en el histograma de intensidades? Si es posible notar diferencias entre los histogramas de ambas imágenes como se muestra a continuación:

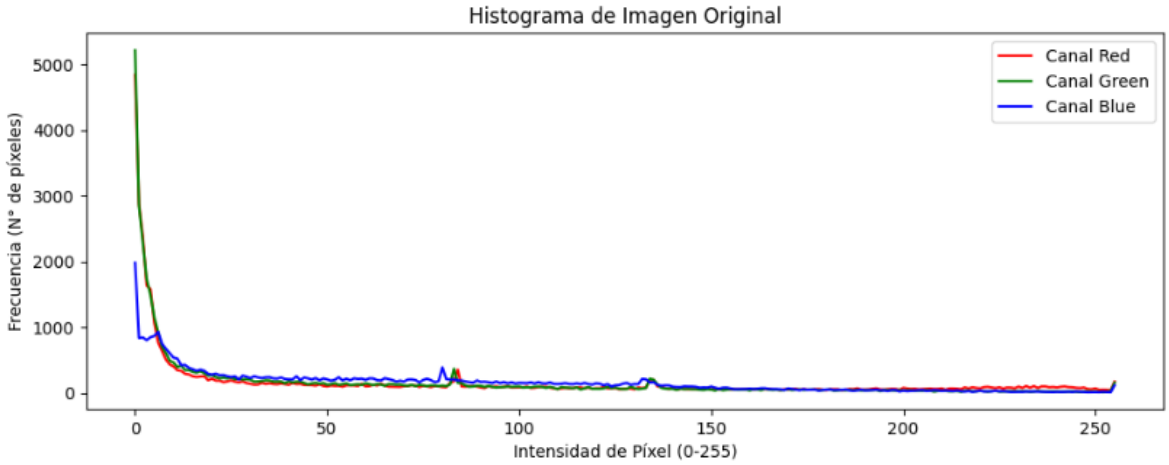

Figura 11. Imagen original

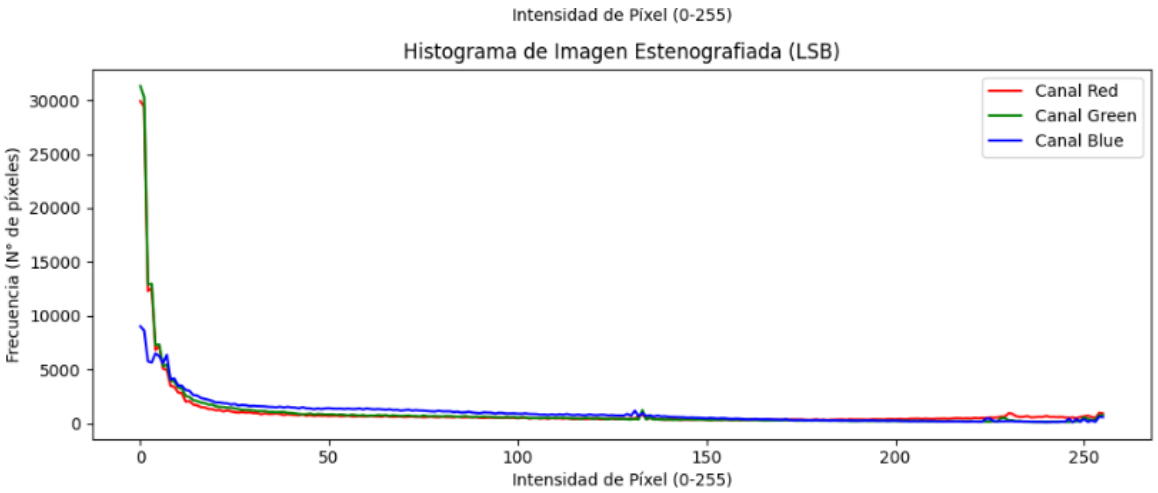

Figura 12. Imagen estenografiada.

Pero ¿por qué apreciamos ese cambio tan abrupto en el número de píxeles (eje y)? Esto sucede por la diferencia entre el procesamiento de una imagen en formato jpg (imagen original) y la conversión a BMP con capacidad máxima, por el formato de la imagen original tenemos colores distribuidos de mejor manera y a la vez están comprimidos, por ello la figura 11 se muestran valores de frecuencia no tan altos, caso contrario al análisis de la imagen estenografiada de la figura 12 en donde al momento de convertirla a

un nuevo formato en capacidad máxima la imagen paso a tener valor específicos de intensidad para ocultar los nuevos bits modificados.

3. Proponga una estrategia para aumentar la capacidad de ocultamiento a 2 LSB por canal.

   Para realizar este proceso podemos hacer un cambio en la máscara de bits donde en lugar de solo variar el último bit se indique el cambio de los últimos bits menos significativos obteniendo una máscara 0xFC, con esto logramos ponemos en cero los últimos dos bytes a modificar de la imagen, con esto se logra que los primeros 6 bits mantengan de manera visible los colores de la imagen original. Con esta técnica se logra pasar de una capacidad máx de 98,300 a una capacidad de más de 196,000 bytes. Al realizar esta práctica habrá cambios en los valores de MSE ya que este crecerá mientras que el valor de PSNR decaerá a valores cercanos a 40 aunque se mantendrá por encima del rango como para que los cambios aún puedan ser imperceptibles a simple vista.